前　言

PID 控制是最早发展起来的控制策略之一，由于其算法简单、鲁棒性好及可靠性高，被广泛应用于过程控制和运动控制中，尤其适用于可建立精确数学模型的确定性系统。然而实际工业生产过程往往具有非线性、时变不确定性，难以建立精确的数学模型，应用常规 PID 控制器不能达到理想的控制效果，而且在实际生产现场中，由于受到参数整定方法繁杂的困扰，常规 PID 控制器参数往往整定不良、性能欠佳，对运行工况的适应性很差。

计算机技术和智能控制理论的发展为复杂动态不确定系统的控制提供了新的途径。采用智能控制技术，可设计智能 PID 和进行 PID 的智能整定。

有关智能 PID 控制等新型 PID 控制理论及其工程应用，近年来已有大量的论文发表。作者多年来一直从事智能控制方面的研究和教学工作，为了促进 PID 控制和自动化技术的进步，反映 PID 控制设计与应用中的最新研究成果，并使广大工程技术人员了解、掌握和应用这一领域的最新技术，学会用 MATLAB 语言进行 PID 控制器的设计，作者编写了这本书，以抛砖引玉，供广大读者学习参考。

本书是在总结作者多年研究成果的基础上，进一步理论化、系统化、规范化、实用化而成的，其特点如下：

（1）PID 控制算法取材新颖，内容先进，重点置于学科交叉部分的前沿研究和介绍一些有潜力的新思想、新方法和新技术，取材着重于基本概念、基本理论和基本方法。

（2）针对每种 PID 算法给出完整的 MATLAB 仿真程序。这些程序都可以在线运行，并给出程序的说明和仿真结果。具有很强的可读性，很容易转化为其他各种实用语言。

（3）着重从应用领域角度出发，突出理论联系实际，面向广大工程技术人员，具有很强的工程性和实用性。书中有大量应用实例及其结果分析，为读者提供了有益的借鉴。

（4）所给出的各种 PID 算法完整，程序设计、结构设计力求简单明了，便于自学和进一步开发。

本书共分 11 章。第 1 章介绍连续系统 PID 控制和离散系统数字 PID 控制的几种基本方法，通过仿真和分析进行了说明。第 2 章介绍常用的数字 PID 控制系统，主要包括串级计算机控制系统的 PID 控制、纯滞后控制系统 Dahlin 算法和基于 Smith 预估的 PID 控制。第 3 章阐明专家 PID 和模糊 PID 整定的基本算法和程序设计方法，其中模糊 PID 包括模糊自适应整定 PID 控制和模糊免疫 PID 控制算法，并进行了仿真分析。第 4 章介绍神经 PID 的几种方法，包括单神经网络 PID 的设计、神经网络并行 PID 控制、PID 的几种神经网络整定方法，并通过仿真进行说明。第 5 章引入基于遗传算法的 PID 控制，主要包括基于遗传算法整定的 PID 控制和基于遗传算法摩擦模型参数辨识的 PID 控制。第 6 章介绍多变量 PID 控制的几种方法，主要包括 PID 控制、单神经元 PID 控制和基于 DRNN 神经网络整定的 PID 控制。第 7 章阐述几种先进的 PID 控制算法，包括基于干扰观测器的 PID 鲁棒控制、基于 NCD 优化的非线性 PID 控制、非线性参数整定的 PID 控制、基于重复控制的 PID 高精度控制、基于零相差前馈补偿的 PID 控制和基于卡尔曼滤波的 PID 控制，每种方法都通过仿真程序进行说明。第 8 章介绍灰色 PID 控制算法和仿真方法，包括基于连续系统的灰色 PID 控制和基于离散系统的灰色 PID 控制。第 9 章引入伺服系统的 PID 控制，包括伺服系统在低速摩擦条件下的 PID 控

制、单质量伺服系统 PID 控制和二质量伺服系统 PID 控制，并进行了仿真说明。第 10 章介绍 PID 在机器人控制中的应用实例，包括确定性单臂机械手、不确定性单臂机械手、*N* 关节机器人的 PID 控制。第 11 章阐述 PID 在实时控制中的应用实例，并给出 PID 控制的 MATLAB 程序和相应的 Borland C++语言实时控制程序。

本书是基于 MATLAB 环境下开发的，各个章节的内容具有很强的独立性，读者可以结合自己的方向深入地进行研究。

本书第 2 版在第 1 版的基础上主要增加了以下内容：基于 *S* 函数的连续系统 Simulink 仿真；基于 *S* 函数的离散系统 Simulink 仿真；基于一种离散微分-跟踪器的 PID 控制；基于 Ziegler-Nichols 方法的 PID 整定；基于 Hopfield 神经网络的 PID 控制；模糊 RBF 网络的 PID 整定；实时遗传算法优化的 PID 控制；基于 Anti-windup 的 PID 控制；基于 PD 增益自适应调节的模型参考自适应控制。并增加了新的一章：机器人 PID 控制。针对某些以 M 语言实现的程序进行了 Simulink 设计，并针对第 1 版中的某些错误进行了修改。

北京航空航天大学尔联洁教授在伺服系统设计方面提出了许多宝贵意见，东北大学徐心和教授和薛定宇教授给予了大力支持和帮助，薛定宇教授在 *S* 函数设计和 Simulink 仿真方面给作者提供了很多的指导。

作者在仿真研究中，得到实验室许多同仁的帮助。在神经网络设计方面得到扈宏杰博士的帮助，在遗传算法和零相差设计等方面得到刘强博士的帮助，在灰色系统设计方面得到李水清硕士的帮助，在机器人控制器设计方面得到卢宇硕士的帮助，在 PID 实时控制方面得到刘涛硕士的帮助，在图表制作中得到了邬强硕士的帮助，在此一并表示感谢。

感谢郝瑞霞、郝春霞、刘海荣在本书的撰写及整理工作中给予的帮助。

本书的研究工作得到了国家自然科学基金（编号：69874037）和航空基金（编号：00E51022）的资助。

由于作者水平有限，书中难免存在一些不足和错误之处，欢迎广大读者批评指正。

刘金琨
北京航空航天大学
2004 年 7 月 1 日

目　　录

第 1 章　数字 PID 控制

自从计算机进入控制领域以来，用数字计算机代替模拟计算机调节器组成计算机控制系统，不仅可以用软件实现 PID 控制算法，而且可以利用计算机的逻辑功能，使 PID 控制更加灵活。数字 PID 控制在生产过程中是一种最普遍采用的控制方法，在机电、冶金、机械、化工等行业中获得了广泛的应用。将偏差的比例（P）、积分（I）和微分（D）通过线性组合构成控制量，对被控对象进行控制，故称 PID 控制器。

1.1　PID 控制原理

在模拟控制系统中，控制器最常用的控制规律是 PID 控制。模拟 PID 控制系统原理框图如图 1-1 所示。系统由模拟 PID 控制器和被控对象组成。

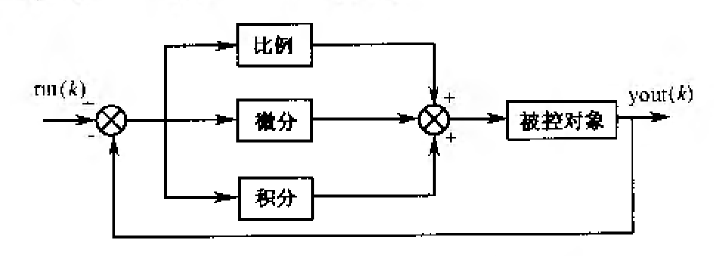


图 1-1　模拟 PID 控制系统原理框图

PID 控制器是一种线性控制器，它根据给定值 rin(t) 与实际输出值 yout(t) 构成控制偏差：

$$\text{error}(t)=\text{rin}(t)-\text{yout}(t) \tag{1.1}$$

PID 的控制规律为：

$$u(t)=k_{\mathrm{p}}\left(\text{error}(t)+\frac{1}{T_{\mathrm{I}}}\int_0^t \text{error}(t)\mathrm{d}t+\frac{T_{\mathrm{D}}\,\mathrm{d}\text{error}(t)}{\mathrm{d}t}\right) \tag{1.2}$$

或写成传递函数的形式：

$$G(s)=\frac{U(s)}{E(s)}=k_{\mathrm{p}}\left(1+\frac{1}{T_{\mathrm{I}}s}+T_{\mathrm{D}}s\right) \tag{1.3}$$

式中，k_{p}——比例系数；T_{I}——积分时间常数；T_{D}——微分时间常数。

简单说来，PID 控制器各校正环节的作用如下：

（1）比例环节：成比例地反映控制系统的偏差信号 error(t)，偏差一旦产生，控制器立即产生控制作用，以减小偏差。

（2）积分环节：主要用于消除静差，提高系统的无差度。积分作用的强弱取决于积分时间常数 T_{I}，T_{I} 越大，积分作用越弱，反之则越强。

（3）微分环节：反映偏差信号的变化趋势（变化速率），并能在偏差信号变得太大之前，在系统中引入一个有效的早期修正信号，从而加快系统的动作速度，减少调节时间。

1.2 连续系统的模拟 PID 仿真

1.2.1 基本的 PID 控制

以二阶线性传递函数为被控对象，进行模拟 PID 控制。在信号发生器中选择正弦信号，仿真时取 $k_p=60$，$k_i=1$，$k_d=3$，输入指令为 $rin(t)=A\sin(2\pi Ft)$，其中 $A=1.0$，$F=0.20\,\text{Hz}$。采用 ODE45 迭代方法，仿真时间为 10s。

仿真方法一

在 Simulink 下进行仿真，PID 控制由 Simulink 下的工具箱提供。

仿真程序：chap1_1.mdl，如图 1-2 所示。

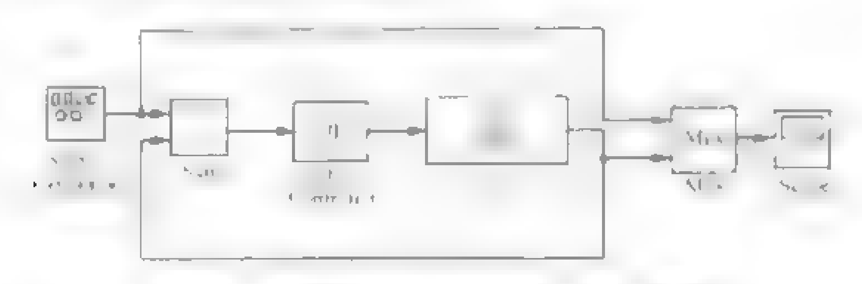


图 1-2　连续系统 PID 的 Simulink 仿真程序

其 PID 控制器采用 Simulink 封装的形式，其内部结构如图 1-3 所示。

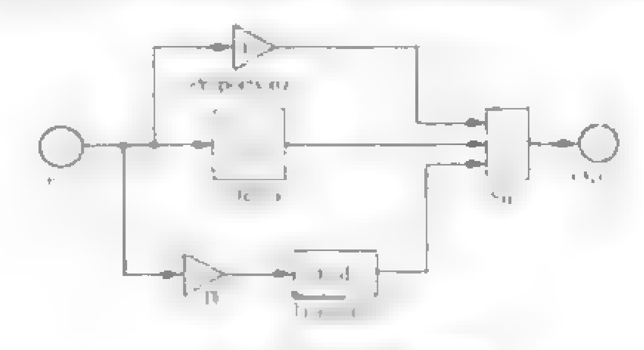


图 1-3　模拟 PID 控制器

连续系统的模拟 PID 控制正弦响应结果如图 1-4 所示。

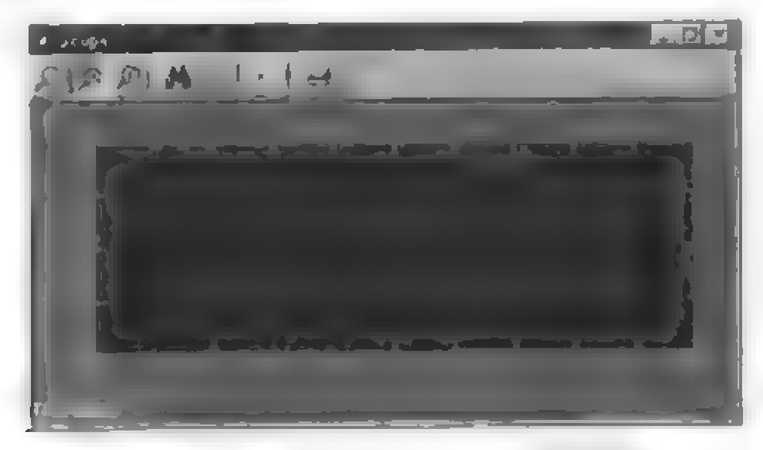


图 1-4　连续系统的模拟 PID 控制正弦响应

仿真方法二

在仿真一的基础上，将仿真结果输出到工作空间中，利用 *M* 函数作图，仿真结果如图 1-5 所示。

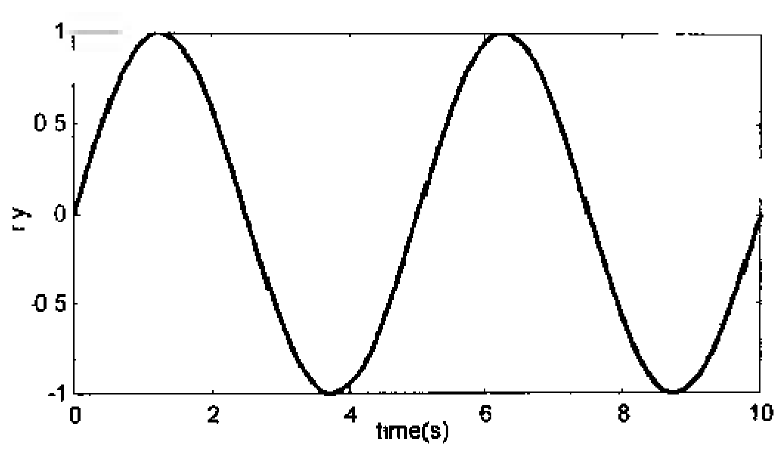


图 1-5　PID 控制正弦响应

仿真程序：chap1_2.mdl。

程序中同时采用了传递函数的另一种表达方式，即状态方程的形式，其中 $A=\begin{bmatrix}0 & 1\\0 & -25\end{bmatrix}$，$B=\begin{bmatrix}0\\133\end{bmatrix}$，$C=\begin{bmatrix}1 & 0\end{bmatrix}$，$D=0$，如图 1-6 所示。

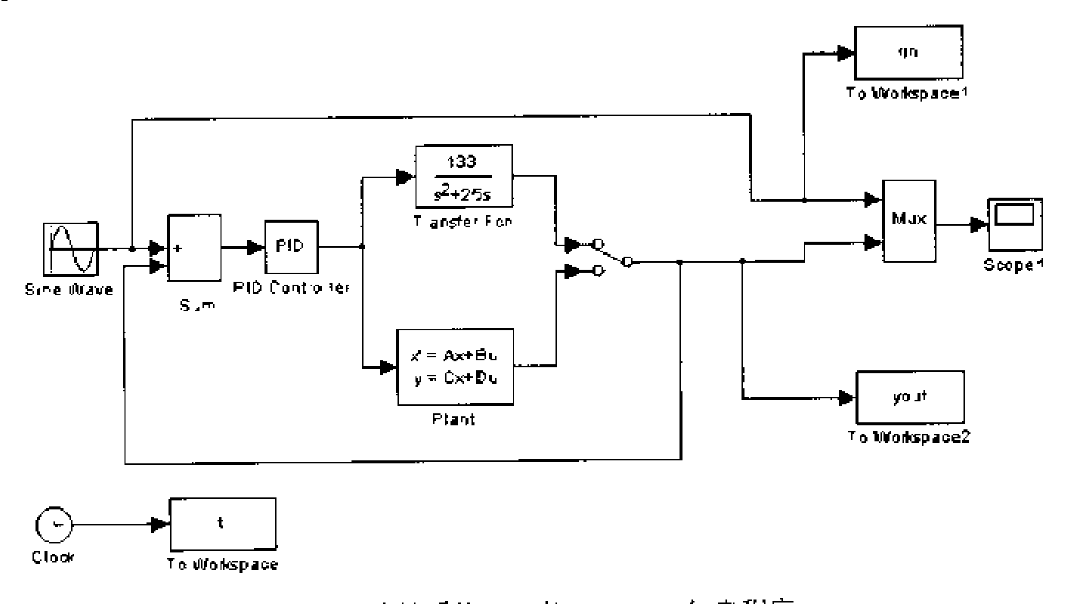


图 1-6　连续系统 PID 的 Simulink 仿真程序

M 函数作图程序：chap1_2plot.m。

```
close all;

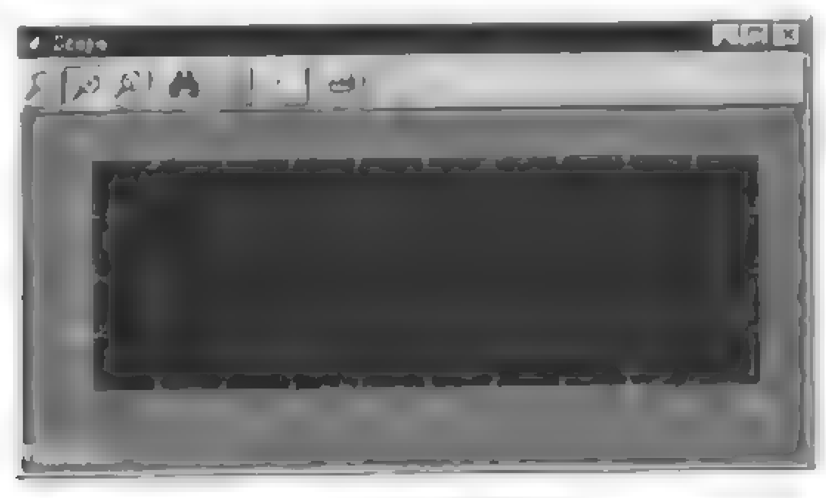
plot(t,rin,'k',t,yout,'k');
xlabel('time(s)');
ylabel('r,y');
```

仿真方法二

S [illegible] Simulink [illegible] Simulink [illegible]被控对象的编程，在仿真[illegible]的基础上，利用 S 函数实现[illegible]的表达、控制器的设计及仿真[illegible]

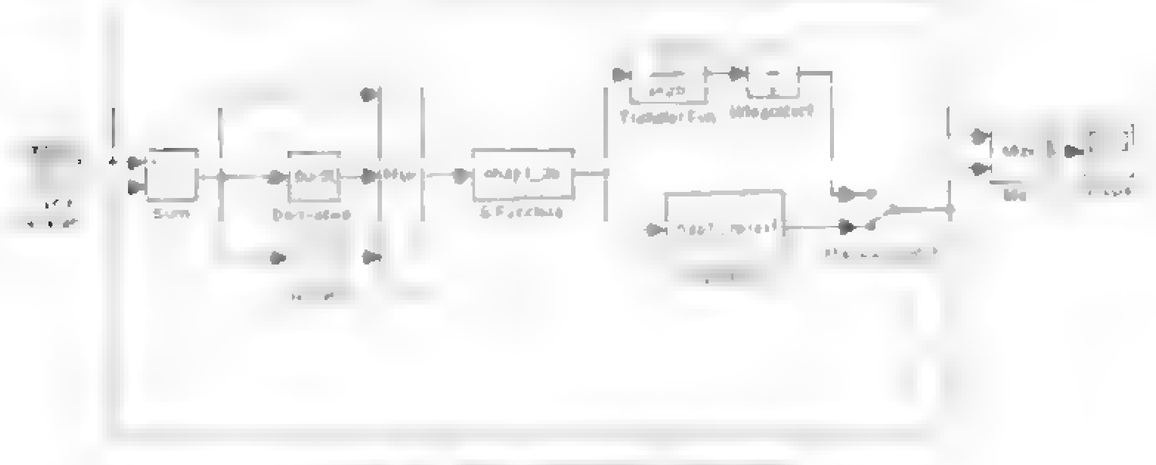
[illegible] mdlInitializeSizes 函数[illegible] mdlDerivatives [illegible] mdlOutputs [illegible] sizes 结构，选择 2 个[illegible] 3 个[illegible] P [illegible] D [illegible] Simulink [illegible]

仿真结果如图 1-7 所示。



[illegible] PID [illegible]

[illegible]：chap1_3.mdl 基于 S 函数的 Simulink [illegible]



[illegible] Simulink [illegible]

S 函数控制器程序：chap1_3s.m

```
    [sys,x0,str,ts]=mdlInitializeSizes;
%Outputs
  case 3,
    sys=mdlOutputs(t,x,u);
%Unhandled flags
  case {2, 4, 9 }
    sys = [];
%Unexpected flags
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end

%mdlInitializeSizes
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;

sys=simsizes(sizes);
x0=[];
str=[];
ts=[];

function sys=mdlOutputs(t,x,u)
error=u(1);
derror=u(2);
errori=u(3);

kp=60;
ki=1;
kd=3;
ut=kp*error+kd*derror+ki*errori;

sys(1)=ut;
```

S **函数被控对象程序：chap1_3plant.m。**

```
%S-function for continuous state equation
```

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)

switch flag,
%Initialization
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
  case 1,
    sys=mdlDerivatives(t,x,u);
%Outputs
  case 3,
    sys=mdlOutputs(t,x,u);
%Unhandled flags
  case {2, 4, 9 }
    sys = [];
%Unexpected flags
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end

%mdlInitializeSizes
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 2;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;

sys=simsizes(sizes);
x0=[0,0];
str=[];
ts=[];

function sys=mdlDerivatives(t,x,u)

sys(1)=x(2);
%sys(2)=-(25+5*sin(t))*x(2)+(133+10*sin(t))*u;
sys(2)=-(25+10*rands(1))*x(2)+(133+30*rands(1))*u;
```

1.2.2 线性时变系统的PID控制

仿真实例

设被控制对象为：

$$G(s)=\frac{K}{s^2+Js}$$

输入指令为 $r(t)=0.5\sin(2\pi t)$，$J=20+10\sin(6\pi t)$，$K=400+300\sin(2\pi t)$，采用PID控制算法进行正弦跟踪。

仿真方法一

采用Simulink仿真，通过Simulink模块实现控制器的设计，取 $k_p=10$，$k_i=0$，$k_d=1.0$。仿真结果如图1-9所示。



图1-9 正弦跟踪

仿真程序：chap1_4.mdl，如图1-10和图1-11所示。

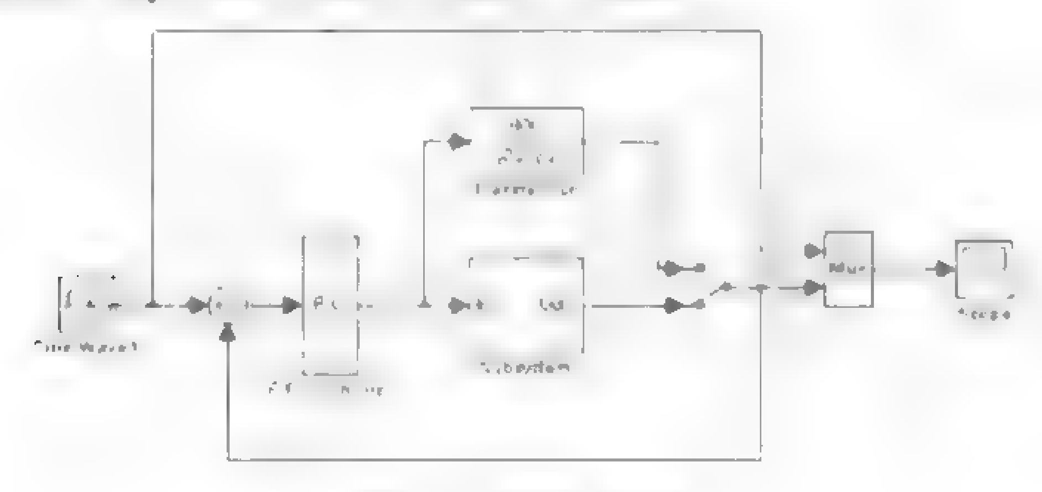


图1-10 Simulink主程序

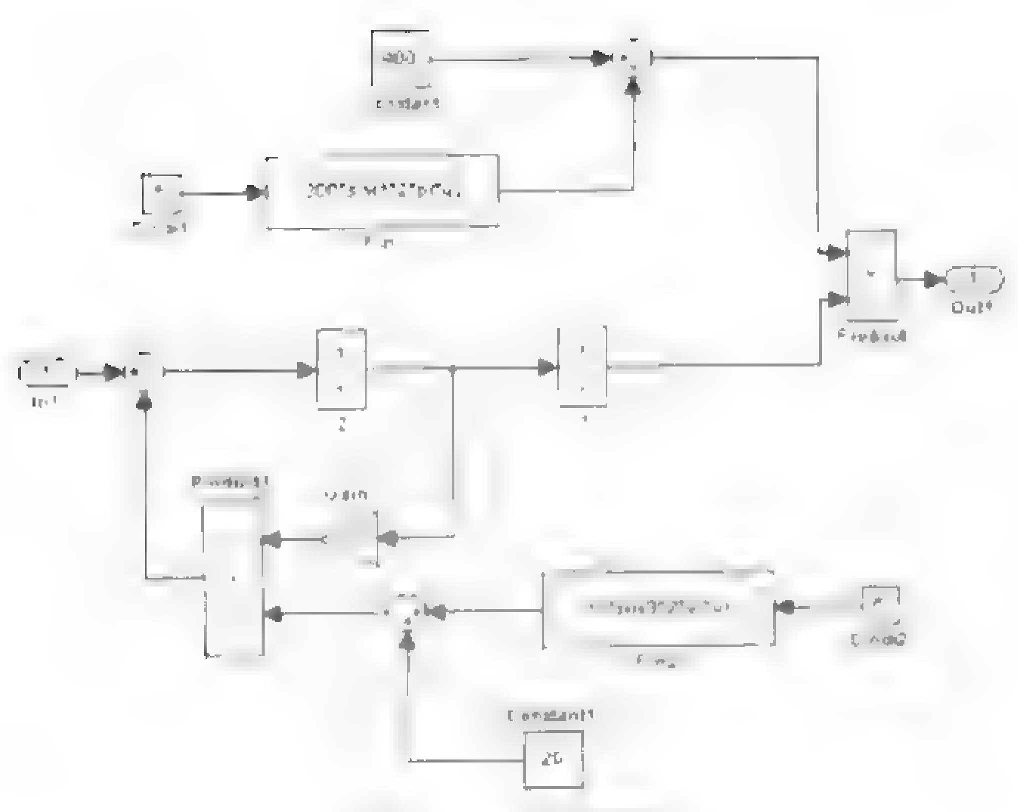


图 1-11 Simulink 子程序

仿真方法二

采用 S 函数的方法进行仿真。不确定对象的表示、控制器的实现及输出由 S 函数完成。在 S 函数中，采用初始化、微分函数和输出函数，即 mdlInitializeSizes 函数、mdlDerivatives 函数和 mdlOutputs 函数。在初始化中采用 sizes 结构，选择 1 个输出，3 个输入。3 个输入实现了 P、I、D 三项的输入。S 函数嵌入在 Simulink 程序中。系统初始状态为：$x(0)=0, \dot{x}(0)=0$，取 $k_p=10$，$k_i=2$，$k_d=1$，仿真结果如图 1-12 所示。

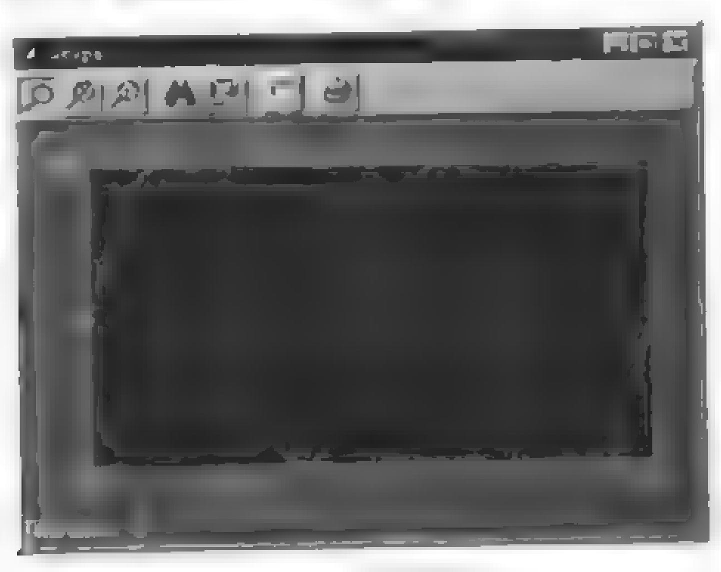


图 1-12 正弦跟踪

仿真程序：chap1_5.mdl，如图 1-13 所示

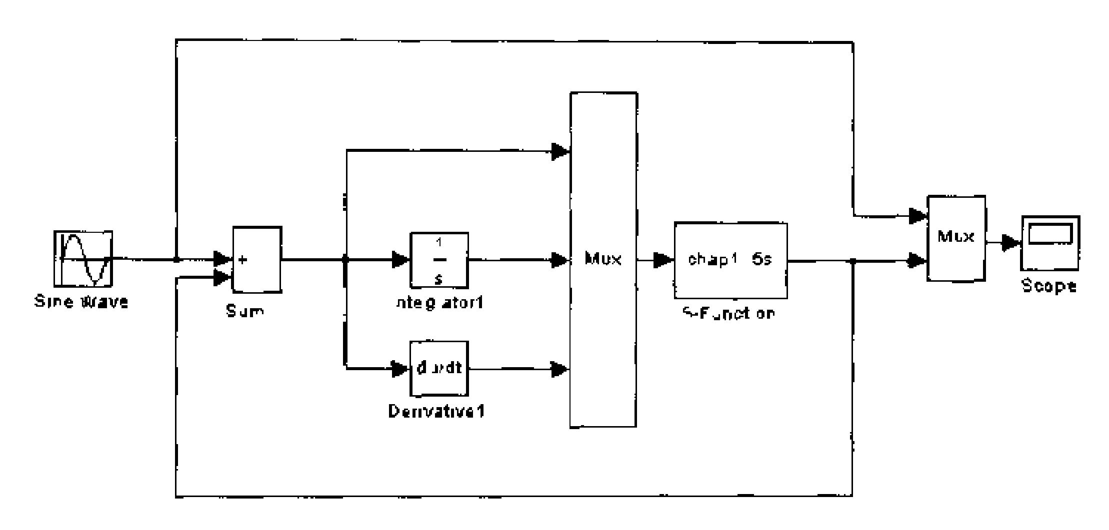


图 1.13 Simulink 子程序

***S* 函数子程序：chap1_5s.m。**

```
function [sys,x0,str,ts]=spacemodel(t,x,u,flag)

switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end

function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 2;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 3;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1; % At least one sample time is needed
sys = simsizes(sizes);
x0  = [0;0];
str = [];
ts  = [0 0];
```

```
function sys=mdlDerivatives(t,x,u)  %Time-varying model
kp=[illegible];
ki=2;
kd=1;

ut=kp*u(1)+ki*u(2)+kd*u(3);

J=20+10*sin(6*pi*t);
K=400+300*sin(2*pi*t);

sys(1)=x(2);
sys(2)=-J*x(2)+K*ut;

function sys=mdlOutputs(t,x,u)

sys(1)=x(1);
```

通过本实例的仿真，可见采用 S 函数，很容易地表示复杂的被控对象及控制算法，特别适合于复杂系统的仿真。

1.3 数字 PID 控制

计算机控制是一种采样控制，它只能根据采样时刻的偏差值计算控制量。因此，连续 PID 控制算法不能直接使用，需要采用离散化方法。在计算机 PID 控制中，使用的是数字 PID 控制器。

1.3.1 位置式 PID 控制算法

按模拟 PID 控制算法，以一系列的采样时刻点 kT 代表连续时间 t，以矩形法数值积分近似代替积分，以一阶后向差分近似代替微分，即：

$$\begin{cases} t \approx kT \quad (k=0,1,2,\cdots) \\ \int_0^t \text{error}(t)\text{d}t \approx T\sum_{j=0}^{k}\text{error}(jT)=T\sum_{j=0}^{k}\text{error}(j) \\ \dfrac{\text{derror}(t)}{\text{d}t} \approx \dfrac{\text{error}(kT)-\text{error}((k-1)T)}{T}=\dfrac{\text{error}(k)-\text{error}(k-1)}{T} \end{cases} \tag{1.4}$$

可得离散 PID 表达式：

$$\begin{aligned} u(k) &= k_{\text{p}}\left(\text{error}(k)+\frac{T}{T_{\text{I}}}\sum_{j=0}^{k}\text{error}(j)+\frac{T_{\text{D}}}{T}(\text{error}(k)-\text{error}(k-1))\right) \\ &= k_{\text{p}}\text{error}(k)+k_{\text{i}}\sum_{j=0}^{k}\text{error}(j)T+k_{\text{d}}\frac{\text{error}(k)-\text{error}(k-1)}{T} \end{aligned} \tag{1.5}$$

式中，$k_i=\frac{k_p}{T_I}$，$k_d=k_pT_D$，T 为采样周期，k 为采样序号，$k=1,2,\cdots$，error($k-1$) 和 error(k) 分别为第 ($k-1$) 和第 k 时刻所得的偏差信号。

位置式 PID 控制系统如图 1-14 所示。

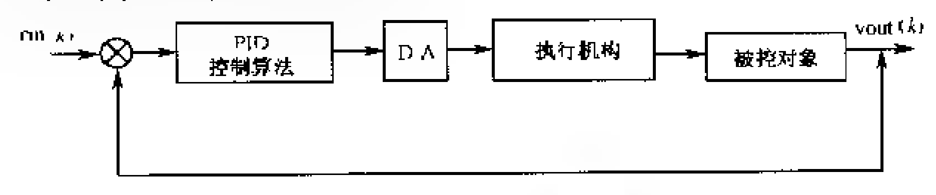


图 1-14 位置式 PID 控制系统

根据位置式 PID 控制算法得到其程序框图如图 1-15 所示。

在仿真过程中，可根据实际情况，对控制器的输出进行限幅：[-10，+10]。

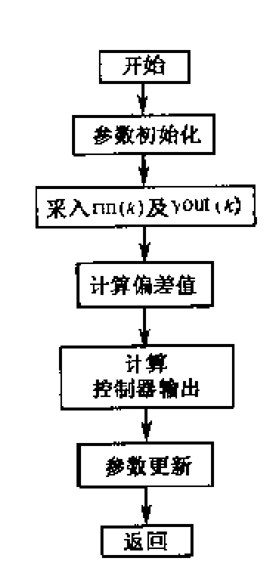


图 1-15 位置式 PID 控制算法程序框图

1.3.2 连续系统的数字 PID 控制仿真

本方法可实现 D/A 及 A/D 的功能，符合数字实时控制的真实情况，计算机及 DSP 的实时 PID 控制都属于这种情况。

仿真方法一

采用 MATLAB 语句形式进行仿真。被控对象为一个电机模型传递函数：

$$G(s)=\frac{1}{Js^2+Bs}$$

式中，$J=0.0067, B=0.10$。

采用 M 函数的形式，利用 ODE45 的方法求解连续对象方程，输入指令信号为 $\text{rin}(k)=0.50\sin(2\pi t)$，采用 PID 控制方法设计控制器，其中 $k_p=20.0, k_d=0.50$。PID 正弦跟踪结果如图 1-16 所示。

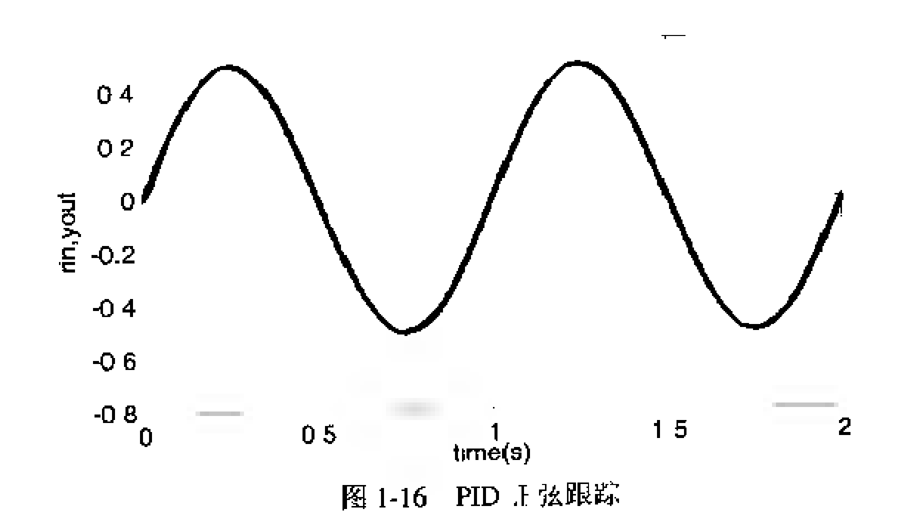


图 1-16 PID 正弦跟踪

控制主程序：chap1_6.m。

```
%Discrete PID control for continuous plant
clear all;
close all;

ts=0.001;  %Sampling time
xk=zeros(2,1);
e_1=0;
u_1=0;

for k=1:1:2000
time(k) = k*ts;

rin(k)=0.50*sin(1*2*pi*k*ts);

para=u_1;              % D/A
tSpan=[0 ts];
[tt,xx]=ode45('chap1_6f',tSpan,xk,[],para);
xk = xx(length(xx),:);    % A/D
yout(k)=xk(1);

e(k)=rin(k)-yout(k);
de(k)=(e(k)-e_1)/ts;

u(k)=20.0*e(k)+0.50*de(k);
%Control limit
if u(k)>10.0
   u(k)=10.0;
end
if u(k)<-10.0
   u(k)=-10.0;
end

u_1=u(k);
e_1=e(k);
end
figure(1);
plot(time,rin,'r',time,yout,'b');
xlabel('time(s)'),ylabel('rin,yout');
figure(2);
plot(time,rin-yout,'r');
xlabel('time(s)'),ylabel('error');
```

连续对象子程序：chap1_6f.m。

```
function dy=PlantModel(t,y,flag,para)
u=para;
J=0.0067;B=0.1;

dy=zeros(2,1);
dy(1)=y(2);
dy(2)=-(B/J)*y(2)+(1/J)*u;
```

仿真方法二

采用 Simulink 进行仿真。被控对象为三阶传递函数，采用 Simulink 模块与 *M* 函数相结合的形式，利用 ODE45 的方法求解连续对象方程，主程序由 Simulink 模块实现，控制器由 *M* 函数实现。输入指令信号为一个采样周期 1ms 的正弦信号。采用 PID 方法设计控制器，其中 $k_p=1.5, k_i=2.0, k_d=0.05$。误差的初始化是通过时钟功能实现的，从而在 *M* 函数中实现了误差的积分和微分。

控制主程序：chap1_7.mdl，如图 1-17 所示。

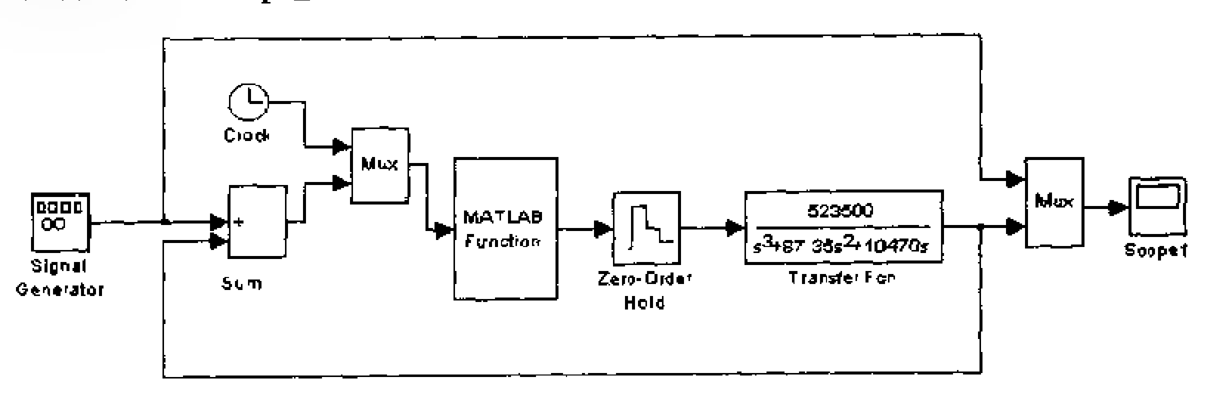


图 1-17　Simulink 仿真程序图

控制器子程序：chap1_7f.m。

```
function [u]=pidsimf(u1,u2)
persistent pidmat error1 error_1

if u1==0
  errori=0
  error_1=0
end

ts=0.001;
kp=1.5;
ki=2.0;
kd=0.05;
```

[illegible]

PID [illegible]图 1-18 所示。

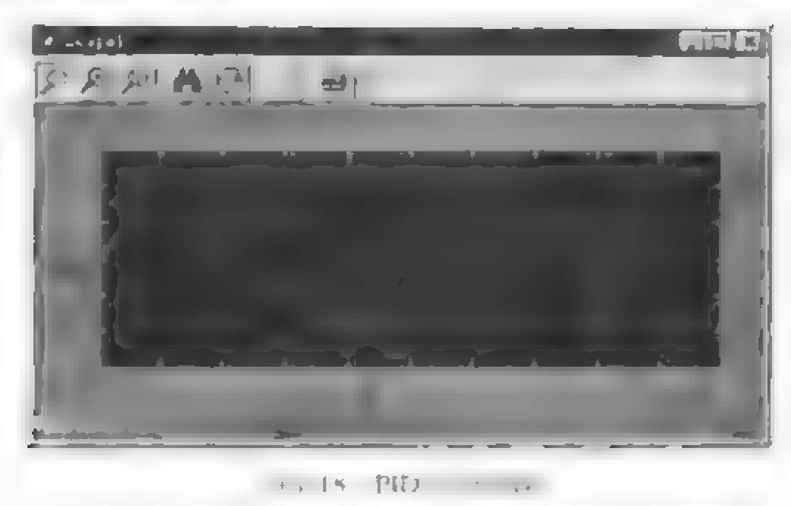


图 1-18 PID [illegible]

仿真方法三

利用 S 函数[illegible]。在 S 函数中，采用初始化、更新函数和输出函数，即 mdlInitializeSizes 函数、mdlUpdates 函数和 mdlOutputs 函数。在初始化中采用 sizes 结构，选择[illegible]个输出，[illegible]个输入，其中[illegible]个输入[illegible]，另一个输入为[illegible]。[illegible] S 函数[illegible] Simulink [illegible]，采样时间为 1ms。

[illegible] Simulink [illegible] chap1_8.mdl，[illegible]图 1-19 所示。

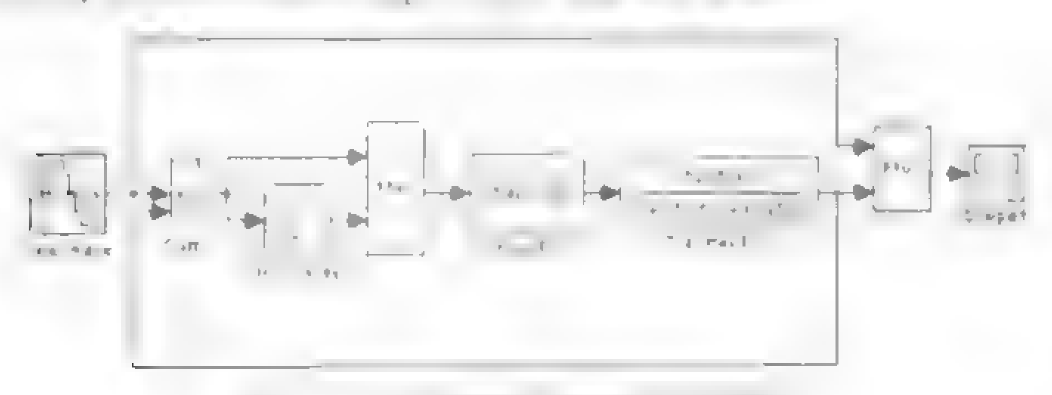


图 1-19 基于 S 函数的 Simulink 仿真程序图

S 函数仿真程序：chap1_8s.m

[illegible]

```
    sys   mdlUpdates(x u ,
case 3            % computation of control signal
%     sys - mdlOutputs t,x,u,kp,ki,kd,MTab ,
    sys mdlOutputs(t,x,u);
case {1, 4, 9    % unused flag values
    sys - [];
otherwise         % error handling
    error [ Unhandled flag -   ,num2str(flag)],;
end;

% ---
% when flag-0, perform system initialization
% ---
function [sys,x0,str,ts] - mdlInitializeSizes
sizes   simsizes;        % read default control variables
sizes.NumContStates  0; % no continuous states
sizes.NumDiscStates - 3; % 3 states and assume they are the P,I/D components
sizes.NumOutputs   1;    % 2 output variables: control u(t) and state x(s)
sizes.NumInputs   ;      % 4 input signals
sizes.DirFeedthrough   1;% input reflected directly in output
sizes.NumSampleTimes  1;% single sampling period
sys   simsizes(sizes ;   %
x0  [0, 0; 0];          % zero initial states
str  [];
ts    1 0];              % sampling period
% ---
% when flag-2, updates the discrete states
% ---
function sys   mdlUpdates x, 
T   1;
sys [ u 1 ;
     x(2,+u 1)*T;
      (1  (2 , T];

% ---
% when flag-3, computates the output signals
% ---
function sys   mdlOutputs(t,x,u,kp,ki,kd,MTab)

kp 1.5;
ki 2.0;
kd 0.05;
```

仿真结果如图1-20所示。



图1-20　PID正弦跟踪

1.3.3　离散系统的数字PID控制仿真

仿真实例

设被控制对象为：

$$G(s)=\frac{523500}{s^3+87.35s^2+10470s}$$

采样时间为1ms，采用Z变换进行离散化，经过Z变换后的离散化对象为：

$$\begin{aligned}yout(k)=&-den(2)yout(k-1)-den(3)yout(k-2)-den(4)yout(k-3)+\\&num(2)u(k-1)+num(3)u(k-2)+num(4)u(k-3)\end{aligned}$$

仿真方法一

针对离散系统的阶跃信号、正弦信号和方波信号的位置响应，设计离散PID控制器。其中，S为信号选择变量，$S=1$时为阶跃跟踪，$S=2$时为方波跟踪，$S=3$时为正弦跟踪。PID阶跃跟踪结果如图1-21～图1-23所示。

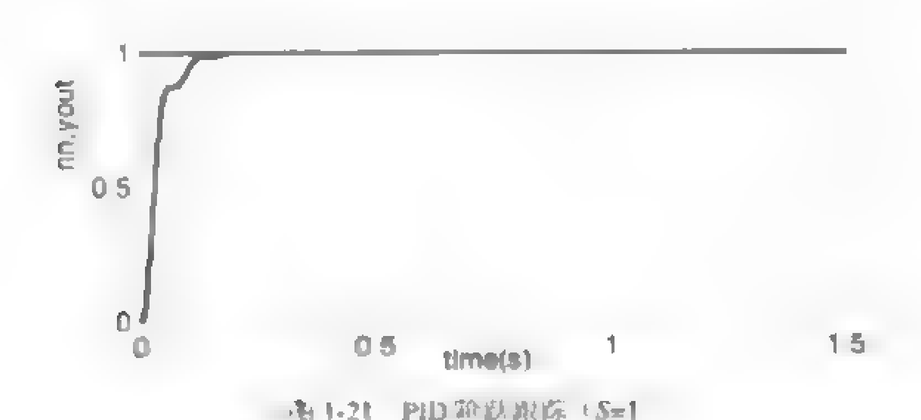


图1-21　PID阶跃跟踪（$S=1$）

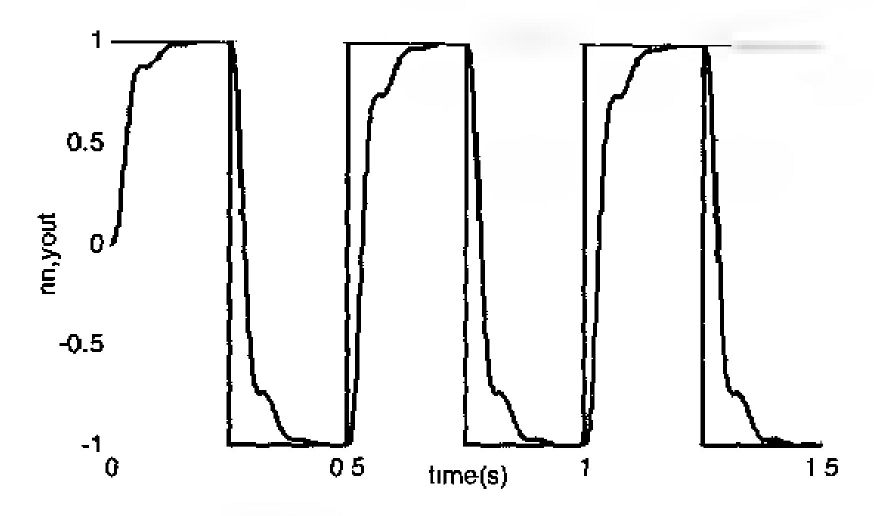


图1-22 PID方波跟踪（S=2）

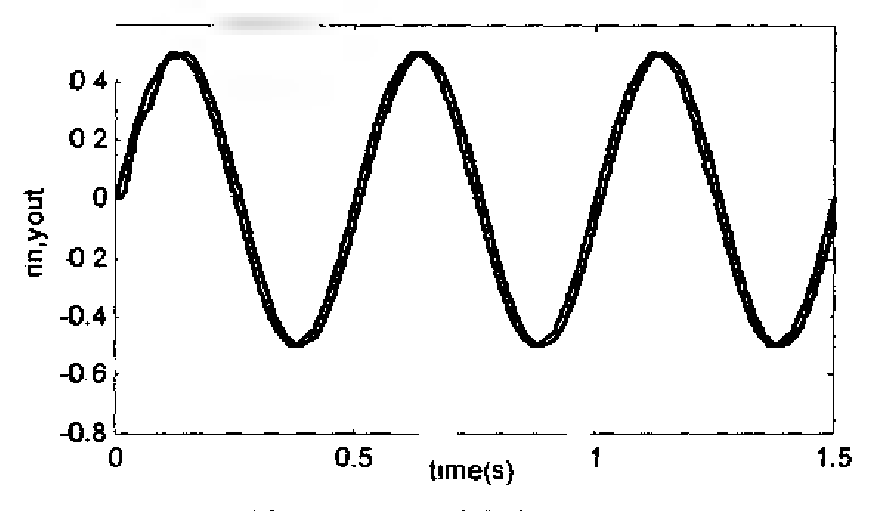


图1-23 PID正弦跟踪（S=3）

仿真程序：chap1_9.m。

```
%PID Controller
clear all;
close all;

ts=0.001;
sys=tf(5.235e005,[1,87.35,1.047e004,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

u_1=0.0;u_2=0.0;u_3=0.0;
y_1=0.0;y_2=0.0;y_3=0.0;
x=[0,0,0]';
error_1=0;
for k=1:1:1500
time(k)=k*ts;
```

```
S=3;
if S==1
   kp=0.50;ki=0.001;kd=0.001;
   rin(k)=1;                          %Step Signal
elseif S==2
   kp=0.50;ki=0.001;kd=0.001;
   rin(k)=sign(sin(2*2*pi*k*ts)); %Square Wave Signal
elseif S==3
   kp=1.5;ki=1.0;kd=0.01;          %Sine Signal
   rin(k)=0.5*sin(2*2*pi*k*ts);
end

u(k)=kp*x(1)+kd*x(2)+ki*x(3);    %PID Controller
%Restricting the output of controller
if u(k)>=10
   u(k)=10;
end
if u(k)<=-10
   u(k)=-10;
end
%Linear model
yout(k)=-den(2)*y_1-den(3)*y_2-den(4)*y_3+num(2)*u_1+num(3)*u_2+num(4)*u_3;

error(k)=rin(k)-yout(k);

%Return of parameters
u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);

x(1)=error(k);                  %Calculating P
x(2)=(error(k)-error_1)/ts;  %Calculating D
x(3)=x(3)+error(k)*ts;        %Calculating I

error_1=error(k);
end
figure(1);
plot(time,rin,'k',time,yout,'k');
xlabel('time(s)'),ylabel('rin,yout');
```

仿真方法二

针对离散系统的三角波、锯齿波和随机信号位置响应，设计离散 PID 控制器，各信号的跟踪结果如图 1-24～图 1-26 所示，其中 S 代表输入指令信号的类型，通过取余指令 mod 实

现三角波和锯齿波。当 $S=1$ 时为三角波，$S=2$ 时为锯齿波，$S=3$ 时为随机信号。在仿真过程中，如果 $D=1$，则通过 pause 命令实现动态演示仿真。在随机信号跟踪中，对随机信号的变化速率进行了限制。

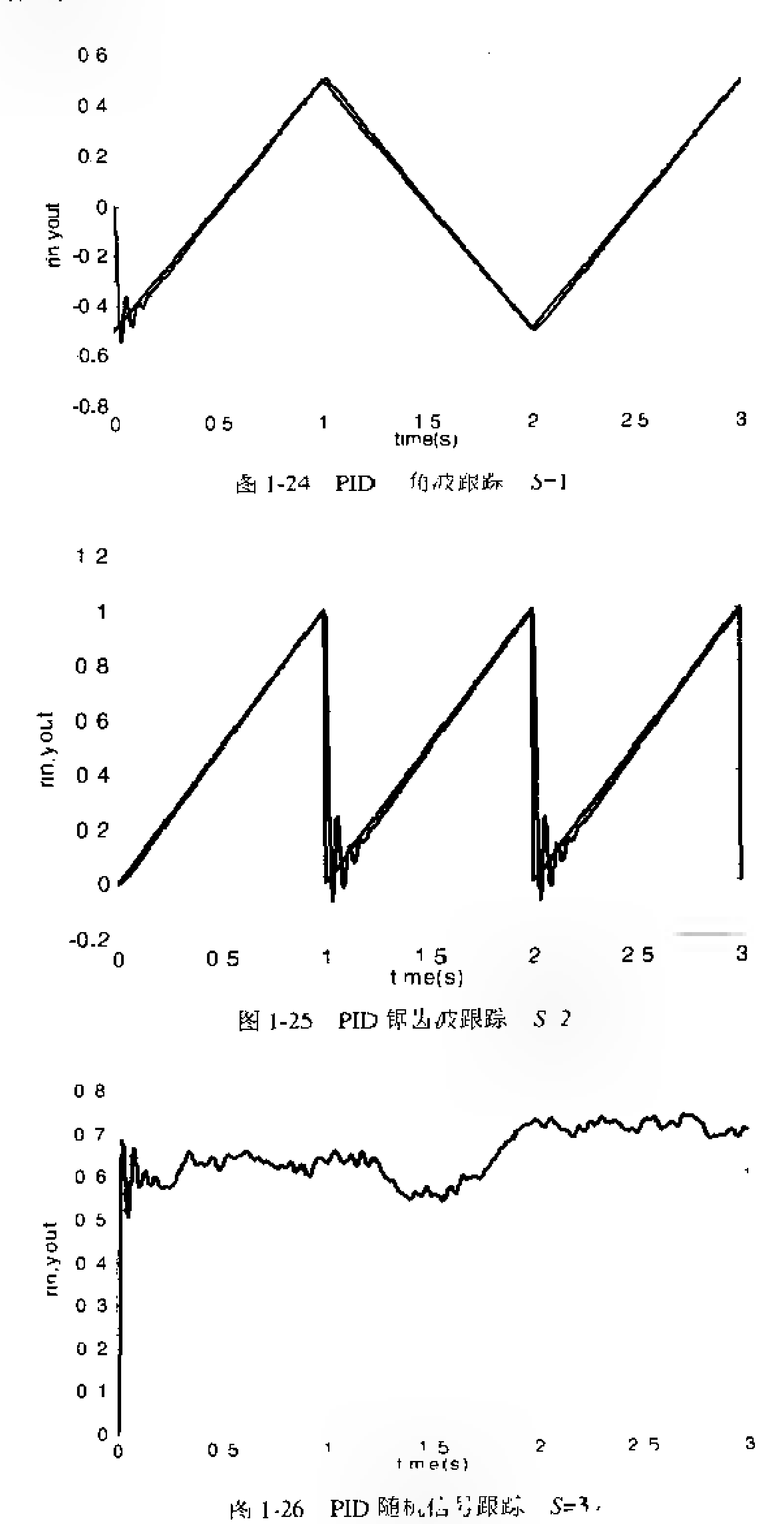


图 1-24　PID 三角波跟踪（$S=1$）

图 1-25　PID 锯齿波跟踪（$S=2$）

图 1-26　PID 随机信号跟踪（$S=3$）

仿真程序：chap1_10.m。

```
%PID Controller
clear all;
close all;

ts=0.001;
sys=tf(5.235e005,[1,87.35,1.047e004,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

u_1=0.0;u_2=0.0;u_3=0.0;
r_1=rand;
y_1=0;y_2=0;y_3=0;

x=[0,0,0]';
error_1=0;

for k=1:1:3000
time(k)=k*ts;

kp=1.0;ki=2.0;kd=0.01;

S=1;
if S==1   %Triangle Signal
   if mod(time(k),2)<1
      rin(k)=mod(time(k),1);
   else
      rin(k)=1-mod(time(k),1);
   end
      rin(k)=rin(k)-0.5;
end
if S==2   %Sawtooth Signal
   rin(k)=mod(time(k),1.0);
end
if S==3   %Random Signal
   rin(k)=rand;
   vr(k)=(rin(k)-r_1)/ts;  %Max speed is 5.0
   while abs(vr(k))>=5.0
   rin(k)=rand;
      vr(k)=abs((rin(k)-r_1)/ts);
```

```
    end
end

u(k)=kp*x(1)+kd*x(2)+ki*x(3);   %PID Controller

%Restricting the output of controller
if u(k)>=10
   u(k)=10;
end
if u(k)<=-10
   u(k)=-10;
end

%Linear model
yout(k)=-den(2)*y_1-den(3)*y_2-den(4)*y_3+num(2)*u_1+num(3)*u_2+num(4)*u_3;
error(k)=rin(k)-yout(k);

r_1=rin(k);

u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);

x(1)=error(k);                %Calculating P
x(2)=(error(k)-error_1)/ts;   %Calculating D
x(3)=x(3)+error(k)*ts;        %Calculating I
xi(k)=x(3);

error_1=error(k);
D=0;
if D==1  %Dynamic Simulation Display
   plot(time,rin,'b',time,yout,'r');
   pause(0.00000000000000000);
end
end
plot(time,rin,'r',time,yout,'b');
xlabel('time(s)');ylabel('rin,yout');
```

上述 PID 控制算法的缺点是，由于采用全量输出，所以每次输出均与过去的状态有关，计算时要对 error(k) 量进行累加，计算机输出控制量 $u(k)$ 对应的是执行机构的实际位置偏差，如果位置传感器出现故障，$u(k)$ 可能会出现大幅度变化。$u(k)$ 的大幅度变化会引起执行机构位置的大幅度变化，这种情况在生产中是不允许的，在某些重要场合还可能造成重大事故。

为避免这种情况的发生，可采用增量式 PID 控制算法

仿真方法三

采用 Simulink 实现离散 PID 控制器的设计，其中离散 PID 控制的封装界面如图 1-27 所示，在该界面中可设定 PID 的三个系数、采样时间及控制输入的上下界。仿真结果如图 1-28 所示

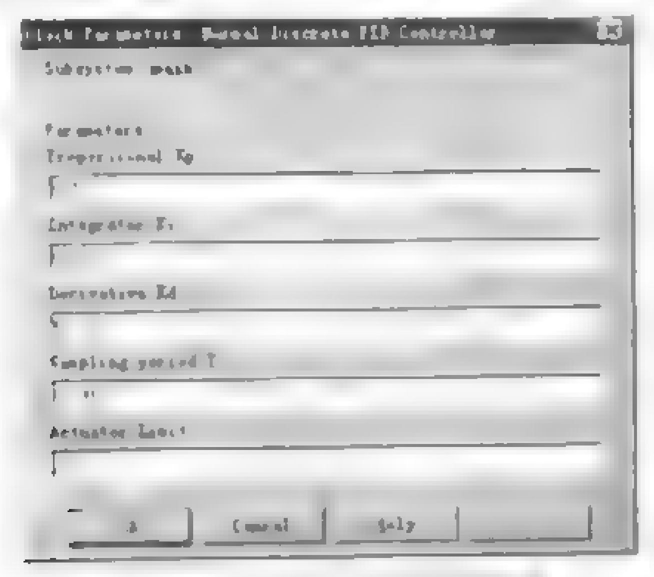


图 1-27 离散 PID 控制器封装界面



图 1-28 [illegible]

仿真程序：chap1_11.mdl，如图 1-29 和图 1-30 所示，其中 PID 控制的比例、微分和积分项分别由 Simulink 模块实现

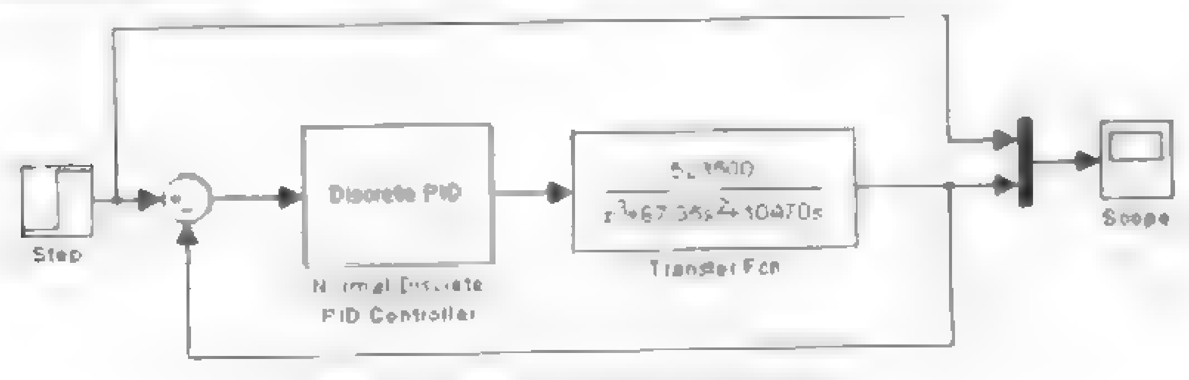


图 1-29 离散 PID 控制的 Simulink 主程序

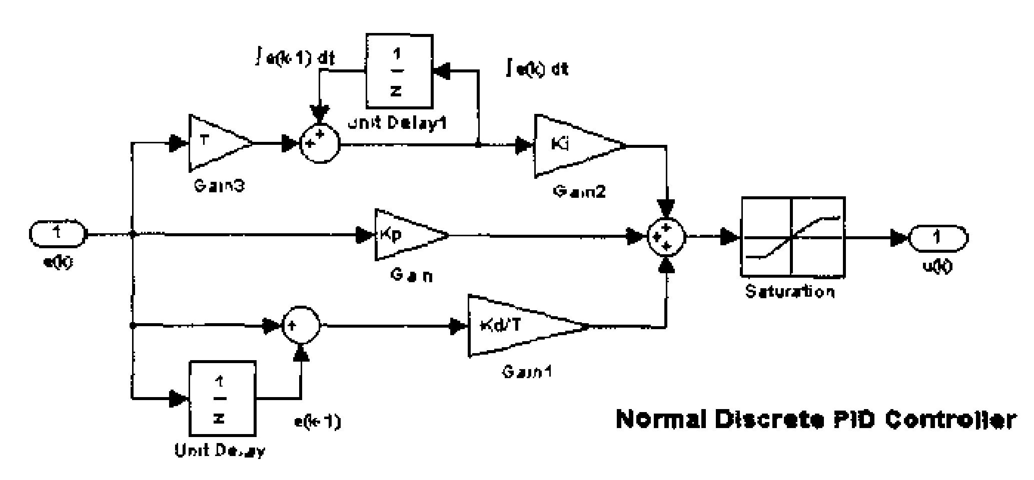


图 1-30　离散 PID 控制的 Simulink 控制器程序

1.3.4　增量式 PID 控制算法及仿真

当执行机构需要的是控制量的增量（例如驱动步进电机）时，应采用增量式 PID 控制。根据递推原理可得：

$$u(k-1)=k_{\mathrm{p}}(\mathrm{error}(k-1)+k_{\mathrm{i}}\sum_{j=0}^{k-1}\mathrm{error}(j)+k_{\mathrm{d}}(\mathrm{error}(k-1)-\mathrm{error}(k-2))) \tag{1.6}$$

增量式 PID 控制算法：

$$\Delta u(k)=u(k)-u(k-1)$$

$$\Delta u(k)=k_{\mathrm{p}}(\mathrm{error}(k)-\mathrm{error}(k-1))+k_{\mathrm{i}}\mathrm{error}(k)+k_{\mathrm{d}}(\mathrm{error}(k)-2\mathrm{error}(k-1)+\mathrm{error}(k-2)) \tag{1.7}$$

根据增量式 PID 控制算法，设计了仿真程序。设被控制对象如下：

$$G(s)=\frac{400}{s^2+50s}$$

PID 控制参数为：$k_{\mathrm{p}}=8, k_{\mathrm{i}}=0.10, k_{\mathrm{d}}=10$。

仿真程序：chap1_12.m。

```
%Increment PID Controller
clear all;
close all;

ts=0.001;
sys=tf(400,[1,50,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

u_1=0.0;u_2=0.0;u_3=0.0;
y_1=0;y_2=0;y_3=0;

x=[0,0,0]';

error_1=0;
error_2=0;
for k=1:1:1000
```

```
    time(k)=k*ts;

    rin(k)=1.0;
    kp=8;
    ki=0.10;
    kd=10;

    du(k)=kp*x(1)+kd*x(2)+ki*x(3);
    u(k)=u_1+du(k);

    if u(k)>=10
       u(k)=10;
    end
    if u(k)<=-10
       u(k)=-10;
    end
    yout(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;

    error=rin(k)-yout(k);
    u_3=u_2;u_2=u_1;u_1=u(k);
    y_3=y_2;y_2=y_1;y_1=yout(k);

    x(1)=error-error_1;             %Calculating P
    x(2)=error-2*error_1+error_2;   %Calculating D
    x(3)=error;                     %Calculating I

    error_2=error_1;
    error_1=error;
end
plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('rin,yout');
```

增量式 PID 阶跃跟踪结果如图 1-31 所示。

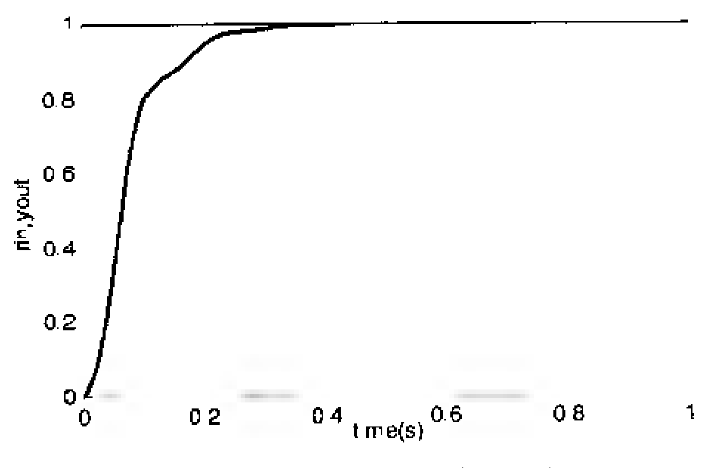


图 1-31　增量式 PID 阶跃跟踪

由于控制算法中不需要累加，控制增量$\Delta u(k)$仅与最近k次的采样有关，所以误动作时影响小，而且较容易通过加权处理获得比较好的控制效果。

在计算机控制系统中，PID控制是通过计算机程序实现的，因此它的灵活性很大。一些原来在模拟PID控制器中无法实现的问题，在引入计算机以后，就可以得到解决，于是产生了一系列的改进算法，形成非标准的控制算法，以改善系统品质，满足不同控制系统的需要。

1.3.5 积分分离PID控制算法及仿真

在普通PID控制中，引入积分环节的目的主要是为了消除静差，提高控制精度。但在过程的启动、结束或大幅度增减设定时，短时间内系统输出有很大的偏差，会造成PID运算的积分积累，致使控制量超过执行机构可能允许的最大动作范围对应的极限控制量，引起系统较大的超调，甚至引起系统较大的振荡，这在生产中是绝对不允许的。

积分分离控制基本思路是，当被控量与设定值偏差较大时，取消积分作用，以免由于积分作用使系统稳定性降低，超调量增大；当被控量接近给定值时，引入积分控制，以便消除静差，提高控制精度。其具体实现步骤如下：

（1）根据实际情况，人为设定阈值$\varepsilon>0$；

（2）当$|error(k)|>\varepsilon$时，采用PD控制，可避免产生过大的超调，又使系统有较快的响应；

（3）当$|error(k)|\leqslant\varepsilon$时，采用PID控制，以保证系统的控制精度。

积分分离控制算法可表示为：

$$u(k)=k_{\mathrm{p}}error(k)+\beta k_{\mathrm{i}}\sum_{j=0}^{k}error(j)T+k_{\mathrm{d}}(error(k)-error(k-1))/T \tag{1.8}$$

式中，T为采样时间，β项为积分项的开关系数

$$\beta=\begin{cases}1 & |error(k)|\leqslant\varepsilon\\ 0 & |error(k)|>\varepsilon\end{cases} \tag{1.9}$$

根据积分分离式PID控制算法得到其程序框图如图1-32所示。

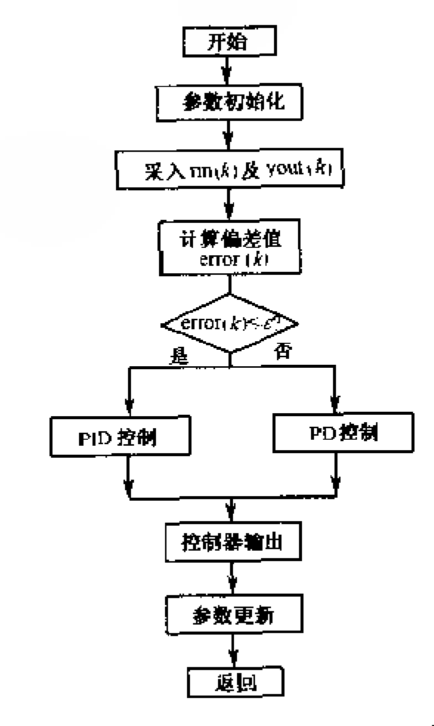


图1-32 积分分离式PID控制算法程序框图

仿真实例

设被控对象为一个延迟对象：

$$G(s)=\frac{e^{-80s}}{60s+1}$$

采样时间为20s，延迟时间为4个采样时间，即80s，被控对象离散化为：

$$y(k)=-den(2)y(k-1)+num(2)u(k-5)$$

仿真方法一

采用M语言进行仿真。取$M=1$，采用积分分离式PID控制器进行阶跃响应，对积分分离式PID控制算法进行改进，采用分段积分分离方式，即根据误差绝对值的不同，采用不同的积分强度。仿真中指令信号为$rin(k)=40$，控制器输出限制在[-110,110]，其阶跃式跟踪结果如

图 1-33 所示。取 $M=2$，采用普通 PID 控制，其阶跃式跟踪结果如图 1-34 所示。

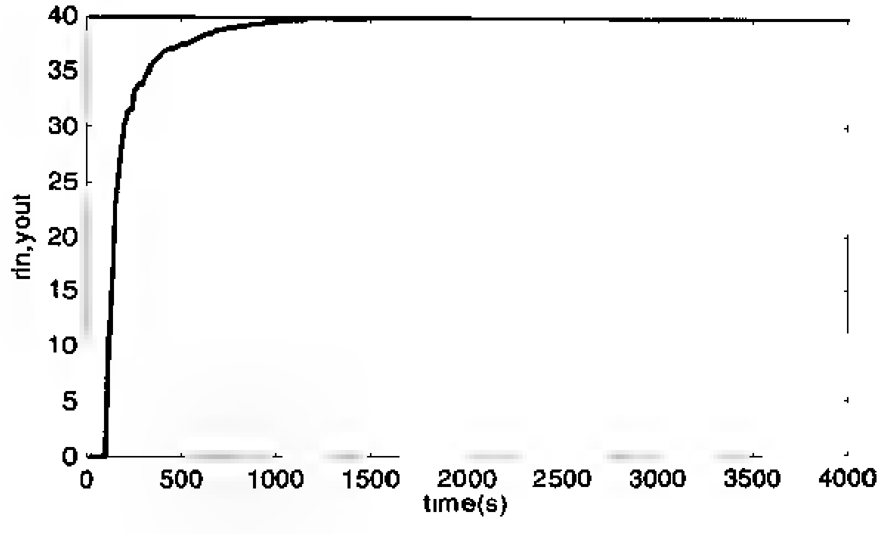


图 1-33　积分分离式 PID 阶跃跟踪（M=1）

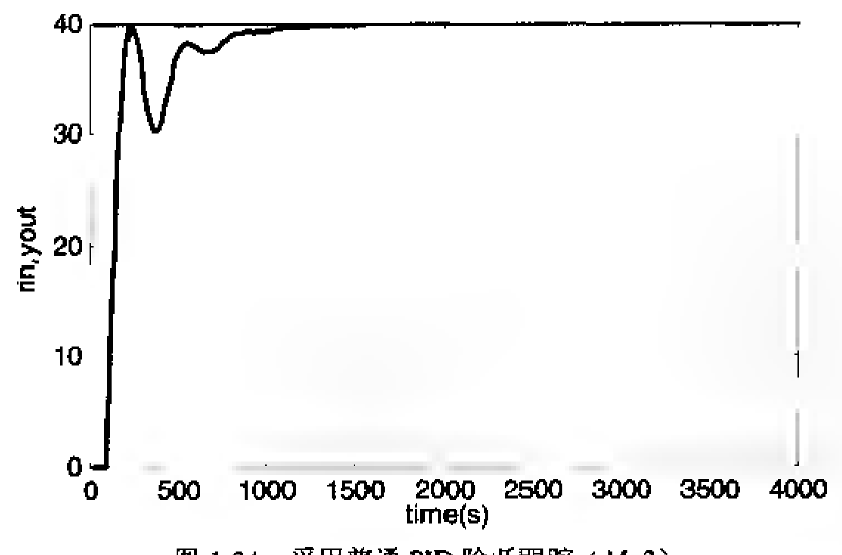


图 1-34　采用普通 PID 阶跃跟踪（M=2）

仿真程序：chap1_13.m。

```
%Integration Separation PID Controller
clear all;
close all;

ts=20;
%Delay plant
sys=tf([1],[60,1],'inputdelay',80);
dsys=c2d(sys,ts,'zoh');
[num,den]=tfdata(dsys,'v');

u_1=0;u_2=0;u_3=0;u_4=0;u_5=0;
```

```
y_1=0;y_2=0;y_3=0;
error_1=0;error_2=0;
ei=0;
for k=1:1:200
time(k)=k*ts;

%Delay plant
yout(k)=-den(2)*y_1+num(2)*u_5;

%I separation
rin(k)=40;
error(k)=rin(k)-yout(k);
ei=ei+error(k)*ts;

M=2;
if M==1            %Using integration separation
   if abs(error(k))>=30&abs(error(k))<=40
      beta=0.3;
   elseif abs(error(k))>=20&abs(error(k))<=30
      beta=0.6;
   elseif abs(error(k))>=10&abs(error(k))<=20
      beta=0.9;
   else
      beta=1.0;
   end
elseif M==2
       beta=1.0;   %Not using integration separation
end

kp=0.80;
ki=0.005;
kd=3.0;
u(k)=kp*error(k)+kd*(error(k)-error_1)/ts+beta*ki*ei;

if u(k)>=110        % Restricting the output of controller
   u(k)=110;
end
if u(k)<=-110
```

由仿真结果可以看出，采用积分分离方法，控制效果有很大的改善。值得注意的是，为保证引入积分作用后系统的稳定性不变，在输入积分作用时比例系数 k_p 可进行相应变化。此外，β 值应根据具体对象及要求而定，若 β 过大，则达不到积分分离的目的；若 β 过小，则会导致无法进入积分区。如果只进行 PD 控制，会使控制出现余差。

仿真方法二

采用 Simulink 仿真，通过 Simulink 模块实现积分分离 PID 控制算法，仿真结果如图 1-35 所示。

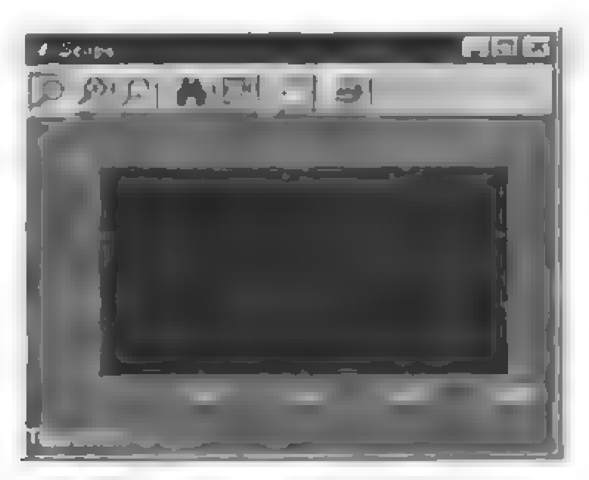


图 1-35 阶跃响应结果

仿真程序的初始化程序：chap1_14f.m

```
dsys=c2d(sys,ts,'zoh');
[num,den]=tfdata(dsys,'v');

kp=1.8;
ki=0.05;
kd=0.20;
```

Simulink 主程序：chap1_14.mdl，如图 1-36 所示。

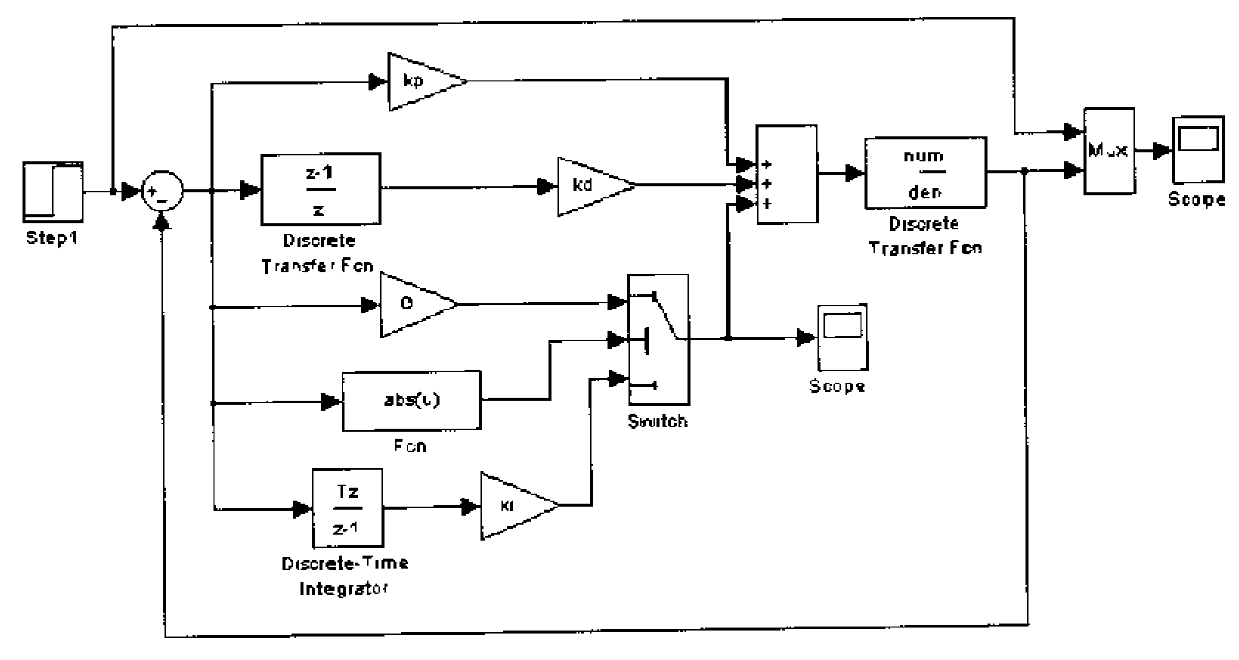


图 1-36　Simulink 仿真程序

1.3.6　抗积分饱和 PID 控制算法及仿真

1．积分饱和现象

所谓积分饱和现象是指若系统存在一个方向的偏差，PID 控制器的输出由于积分作用的不断累加而加大，从而导致执行机构达到极限位置 X_{max}（例如阀门开度达到最大），如图 1-37 所示，若控制器输出 $u(k)$ 继续增大，阀门开度不可能再增大，此时就称计算机输出控制量超出了正常运行范围而进入了饱和区。一旦系统出现反向偏差，$u(k)$ 逐渐从饱和区退出。进入饱和区愈深则退出饱和区所需时间愈长。在这段时间内，执行机构仍停留在极限位置而不能随偏差反向立即做出相应的改变，这时系统就像失去控制一样，造成控制性能恶化。这种现象称为积分饱和现象或积分失控现象。

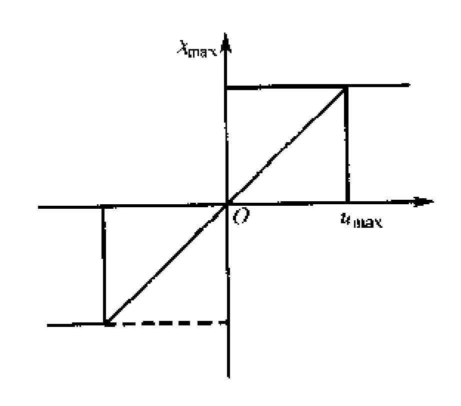


图 1-37　执行机构饱和特性

2．抗积分饱和算法

作为防止积分饱和的方法之一就是抗积分饱和法。该方法的思路是，在计算 $u(k)$ 时，首

先判断上一时刻的控制量 $u(k-1)$ 是否已超出限制范围。若 $u(k-1)>u_{max}$，则只累加负偏差；若 $u(k-1)<u_{max}$，则只累加正偏差。这种算法可以避免控制量长时间停留在饱和区。

仿真实例

设被控制对象为：

$$G(s)=\frac{523500}{s^3+87.35s^2+10470s}$$

采样时间为 1ms，取指令信号 rin$(k)=30$，$M=1$，采用抗积分饱和算法进行离散系统阶跃响应，仿真结果如图 1-38 所示。取 $M=2$，采用普通 PID 算法进行离散系统阶跃响应，其阶跃响应结果如图 1-39 所示。由仿真结果可以看出，采用抗积分饱和 PID 方法，可以避免控制量长时间停留在饱和区，防止系统产生超调。

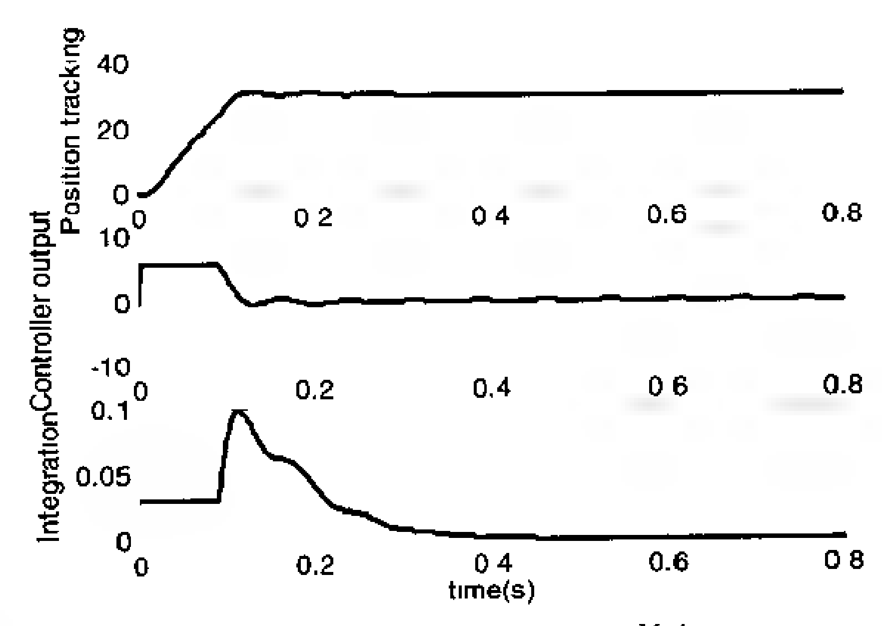


图 1-38　抗积分饱和仿真结果（$M=1$）

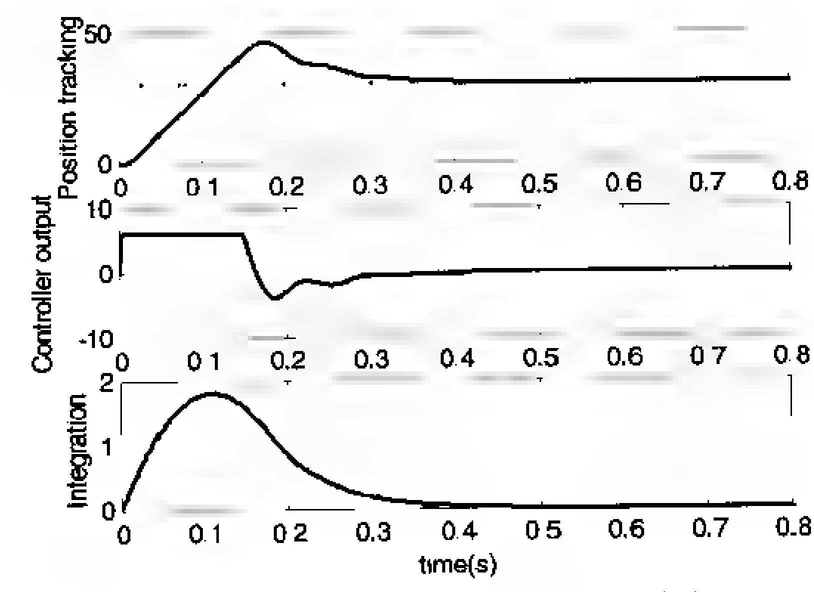


图 1-39　普通 PID 算法进行离散系统阶跃响应结果（$M=2$）

仿真程序：chap1_15.m。

```
%PID Controller with intergration sturation
clear all;
close all;

ts=0.001;
sys=tf(5.235e005,[1,87.35,1.047e004,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

u_1=0.0;u_2=0.0;u_3=0.0;
y_1=0;y_2=0;y_3=0;

x=[0,0,0]';

error_1=0;

um=6;
kp=0.85;ki=9.0;kd=0.0;
rin=30;          %Step Signal

for k=1:1:800
time(k)=k*ts;

u(k)=kp*x(1)+kd*x(2)+ki*x(3);   % PID Controller

if u(k)>=um
   u(k)=um;
end
if u(k)<=-um
   u(k)=-um;
end

%Linear model
yout(k)=-den(2)*y_1-den(3)*y_2-den(4)*y_3+num(2)*u_1+num(3)*u_2+num(4)*u_3;

error(k)=rin-yout(k);

M=2;
if M==1  %Using intergration sturation
```

```
if u(k)>=um
   if error(k)>0
      alpha=0;
   else
      alpha=1;
   end
elseif u(k)<=-um
   if error(k)>0
      alpha=1;
   else
      alpha=0;
   end
else
   alpha=1;
end

elseif M==2   %Not using intergration sturation
    alpha=1;
end

%Return of PID parameters
u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);
error_1=error(k);

x(1)=error(k);                 % Calculating P
x(2)=(error(k)-error_1)/ts;    % Calculating D
x(3)=x(3)+alpha*error(k)*ts;   % Calculating I

xi(k)=x(3);
end
figure(1);
subplot(311);
plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('Position tracking');
subplot(312);
plot(time,u,'r');
xlabel('time(s)');ylabel('Controller output');
subplot(313);
plot(time,xi,'r');
```

```
xlabel('time(s)');ylabel('Integration');
```

1.3.7 梯形积分 PID 控制算法

在 PID 控制律中积分项的作用是消除余差，为了减小余差，应提高积分项的运算精度，为此，可将矩形积分改为梯形积分。梯形积分的计算公式为：

$$\int_0^t e(t)\mathrm{d}t = \sum_{i=0}^{k} \frac{e(i)+e(i-1)}{2}T \tag{1.10}$$

1.3.8 变速积分 PID 算法及仿真

在普通的 PID 控制算法中，由于积分系数 k_i 是常数，所以在整个控制过程中，积分增量不变。而系统对积分项的要求是，系统偏差大时积分作用应减弱甚至全无，而在偏差小时则应加强。积分系数取大了会产生超调，甚至积分饱和，取小了又迟迟不能消除静差。因此，如何根据系统偏差大小改变积分的速度，对于提高系统品质是很重要的。变速积分 PID 可较好地解决这一问题。

变速积分 PID 的基本思想是，设法改变积分项的累加速度，使其与偏差大小相对应：偏差越大，积分越慢；反之则越快。

为此，设置系数 $f(e(k))$，它是 $e(k)$ 的函数。当 $|e(k)|$ 增大时，f 减小，反之增大。变速积分的 PID 积分项表达式为：

$$u_\mathrm{i}(k) = k_\mathrm{i}\left\{\sum_{i=0}^{k-1} e(i) + f[e(k)]e(k)\right\}T \tag{1.11}$$

系数 f 与偏差当前值 $|e(k)|$ 的关系可以是线性的或非线性的，可设为：

$$f[e(k)] = \begin{cases} 1 & |e(k)| \leqslant B \\ \dfrac{A-|e(k)|+B}{A} & B < |e(k)| \leqslant A+B \\ 0 & |e(k)| > A+B \end{cases} \tag{1.12}$$

f 值在[0，1]区间内变化，当偏差 $|e(k)|$ 大于所给分离区间 $A+B$ 后，$f=0$，不再对当前值 $e(k)$ 进行继续累加；当偏差 $|e(k)|$ 小于 B 时，加入当前值 $e(k)$，即积分项变为 $u_\mathrm{i}(k)=k_\mathrm{i}\sum\limits_{i=0}^{k} e(i)T$，与一般 PID 积分项相同，积分动作达到最高速；而当偏差 $|e(k)|$ 在 B 与 $A+B$ 之间时，则累加计入的是部分当前值，其值在 $0\sim|e(k)|$ 之间随 $|e(k)|$ 的大小而变化，因此，其积分速度在 $k_\mathrm{i}\sum\limits_{i=0}^{k-1} e(i)T$ 和 $k_\mathrm{i}\sum\limits_{i=0}^{k} e(i)T$ 之间。变速积分 PID 算法为：

$$u(k) = k_\mathrm{p}e(k) + k_\mathrm{i}\left\{\sum_{i=0}^{k-1} e(i) + f[e(k)]e(k)\right\}\cdot T + k_\mathrm{d}[e(k)-e(k-1)] \tag{1.13}$$

这种算法对 A、B 两参数的要求不精确，参数整定较容易。

仿真实例

设被控对象为一个延迟对象：

$$G(s)=\frac{e^{-80s}}{60s+1}$$

采样时间为 20s，延迟时间为 4 个采样时间，即 80s，取 $k_p=0.45$，$k_d=12$，$k_i=0.0048$，$A=0.4$，$B=0.6$。取 $M=1$，采用变速积分 PID 控制算法进行阶跃响应，其结果如图 1-40 和图 1-41 所示。取 $M=2$，采用普通 PID 控制，其结果如图 1-42 所示。由仿真结果可以看出，变速积分与积分分离两种控制方法很类似，但调节方式不同，前者对积分项采用的是缓慢变化，而后者则采用所谓“开关”控制。变速积分调节质量更高。

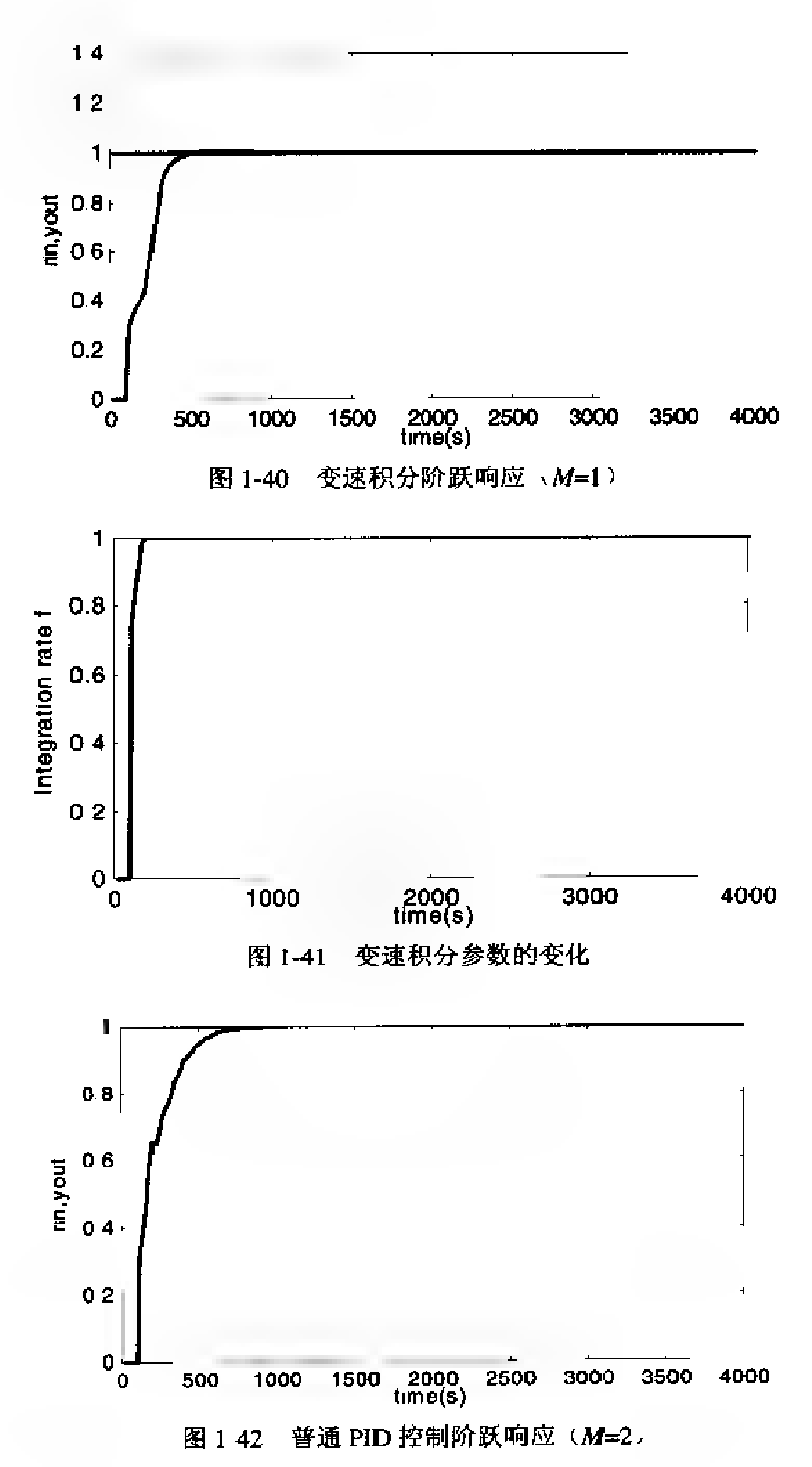


图 1-40　变速积分阶跃响应（M=1）

图 1-41　变速积分参数的变化

图 1-42　普通 PID 控制阶跃响应（M=2）

仿真程序：chap1_16.m。

```
%PID Controller with changing integration rate
clear all;
close all;

%Big time delay Plant
ts=20;
sys=tf([1],[60,1],'inputdelay',80);
dsys=c2d(sys,ts,'zoh');
[num,den]=tfdata(dsys,'v');

u_1=0;u_2=0;u_3=0;u_4=0;u_5=0;
y_1=0;y_2=0;y_3=0;
error_1=0;error_2=0;
ei=0;

for k=1:1:200
time(k)=k*ts;

rin(k)=1.0;  %Step Signal

%Linear model
yout(k)=-den(2)*y_1+num(2)*u_5;
error(k)=rin(k)-yout(k);

kp=0.45;kd=12;ki=0.0048;
A=0.4;B=0.6;

%T type integration
ei=ei+(error(k)+error_1)/2*ts;

M=2;
if M==1      %Changing integration rate
if abs(error(k))<=B
   f(k)=1;
elseif abs(error(k))>B&abs(error(k))<=A+B
   f(k)=(A-abs(error(k))+B)/A;
else
   f(k)=0;
```

```
end

elseif M==2 %Not changing integration rate
    f(k)=1;
end

u(k)=kp*error(k)+kd*(error(k)-error_1)/ts+ki*f(k)*ei;

if u(k)>=10
   u(k)=10;
end
if u(k)<=-10
   u(k)=-10;
end
%Return of PID parameters
u_5=u_4;u_4=u_3;u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);

error_2=error_1;
error_1=error(k);
end
figure(1);
plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,f,'r');
xlabel('time(s)');ylabel('Integration rate f');
```

1.3.9 带滤波器的PID控制仿真

仿真实例一

验证低通滤波器的滤波性能。

设低通滤波器为：

$$Q(s)=\frac{1}{0.04s+1}$$

采样时间为1ms，输入信号为带有高频正弦噪声（100Hz）的低频（0.2Hz）正弦信号。采用低通滤波器滤掉高频正弦信号。滤波器的离散化采用Tustin变换，其Bode图如图1-43和图1-44所示。仿真结果表明，该滤波器对高频信号具有很好的滤波作用。

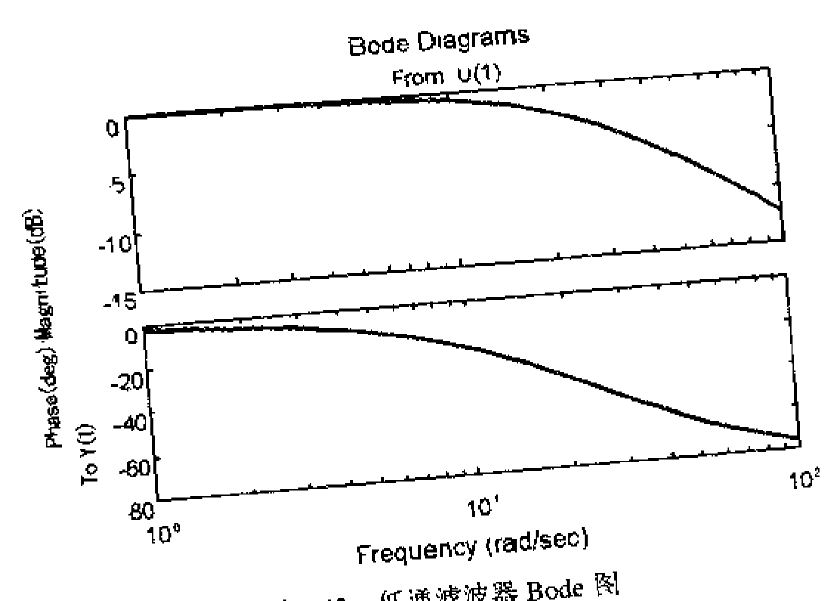


图 1-43　低通滤波器 Bode 图

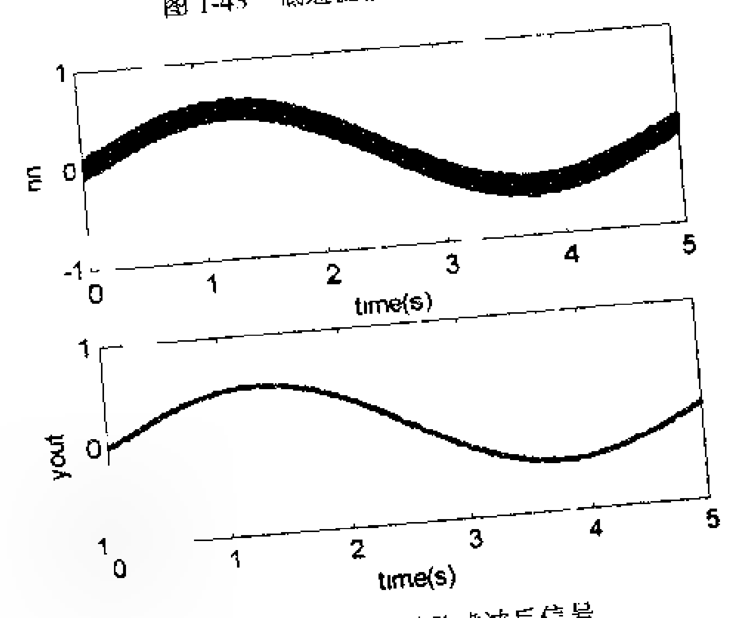


图 1-44　原始信号及滤波后信号

仿真程序：chap1_17.m。

```
%Low Pass Filter
clear all;
close all;

ts=0.001;
Q=tf([1],[0.04,1]);   %Low Freq Signal Filter
Qz=c2d(Q,ts,'tustin');
[num,den]=tfdata(Qz,'v');

y_1=0;y_2=0;
r_1=0;r_2=0;
for k=1:1:5000
time(k)=k*ts;
```

```
%Input Signal with disturbance
D(k)=0.10*sin(100*2*pi*k*ts);  %Disturbance signal
rin(k)=D(k)+0.50*sin(0.2*2*pi*k*ts); %Input Signal

yout(k)=-den(2)*y_1+num(1)*rin(k)+num(2)*r_1;

y_2=y_1;y_1=yout(k);
r_2=r_1;r_1=rin(k);
end
figure(1);bode(Q);
figure(2);
subplot(211);
plot(time,rin,'r');
xlabel('time(s)');ylabel('rin');
subplot(212);
plot(time,yout,'b');
xlabel('time(s)');ylabel('yout');
```

仿真实例二

采用低通滤波器的PID控制。

设被控制对象为三阶传递函数：

$$G_p(s)=\frac{523500}{s^3+87.35s^2+10470s}$$

低通滤波器为：

$$Q(s)=\frac{1}{0.04s+1}$$

采样时间为1ms，干扰信号加在对象的输出端。

仿真方法一

采用M语言进行仿真。分三种情况进行：$M=1$时，为未加干扰信号；$M=2$时，为加干扰信号未加滤波；$M=3$时，为加干扰信号加滤波。阶跃响应结果如图1-45～图1-47所示。

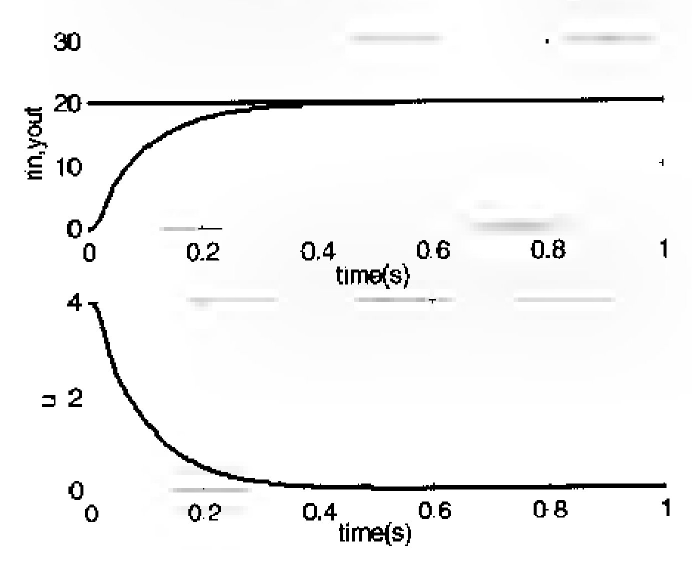


图1-45　普通PID控制阶跃响应（M=1）

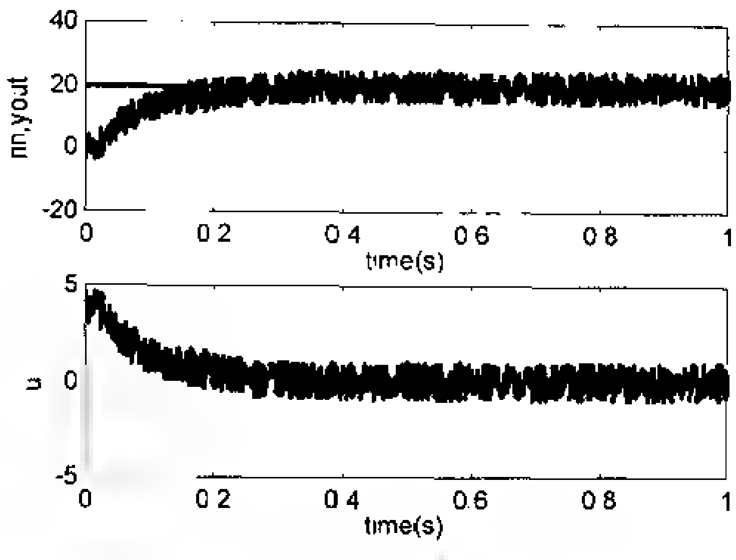


图 1-46　无滤波器时 PID 控制阶跃响应　M=2

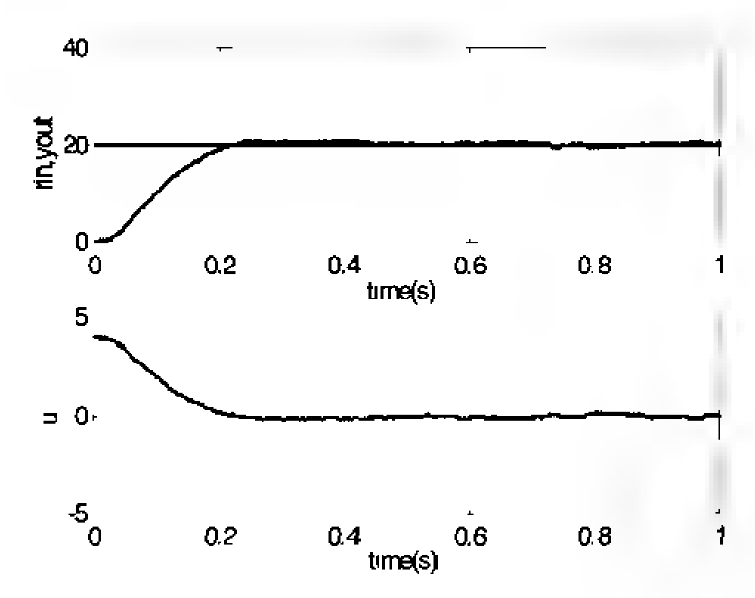


图 1-47　加入滤波器后 PID 控制阶跃响应（M=3）

仿真程序：chap1_18.m。

```
%PID Controller with Partial differential
clear all;
close all;

ts=0.001;
sys=tf(5.235e005,[1,87.35,1.047e004,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

u_1=0;u_2=0;u_3=0;u_4=0;u_5=0;
y_1=0;y_2=0;y_3=0;
```

```
yy 1  -
e.ror . (;error 2 0;e1 0,

kp 0.20;k. 0.C5;

sys1 'f([1],[C.04,1] ;   %Low Freq S.gnal F1 ter
dsys1 c2d sys1,ts, t.st1n' ;
[num1,den1] tfdata(dsys1,'v  ;
f_1-0;

M-1;
for k 1 1:1000
time k) k*ts;

rin(k  2( , %Step Signal

%Linear model
yo. (k   den 2 *y 1 den(3 *y 2 den 4 *y_3+n.m 2)*u 1+...
      n.m 3 *u_2+num(4 *u_3;

if M  1                %No dist.rbance s.gnal
  error(k  r .. k) yo.t(k ,
  filty(k) yo.t k,;
end

D k  5.0*rands(1 ;      %Disturbance signal
yyout(k  yo.t(k +D(k ;

.f M  2                %No f.lter
  filty(k  yyout k ;
  error k) rin k) filty k,;
end
if M- 3                %Using low frequency filter
  f.lty k   den1(2 *f_1+n.m1,1 *(yyout(k)+yy 1 ;
  error k  rin(k) fi.ty k ;
end

%I separation
```

```
if abs(error(k))<=0.8
   ei=ei+error(k)*ts;
else
   ei=0;
end
   u(k)=kp*error(k)+ki*ei;

if u(k)>=10      % Restricting the output of controller
   u(k)=10;
end
if u(k)<=-10
   u(k)=-10;
end
%------------Return of PID parameters------------
rin_1=rin(k);
u_5=u_4;u_4=u_3;u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);

f_1=filty(k);
yy_1=yyout(k);

error_2=error_1;
error_1=error(k);
end
figure(1);
plot(time,rin,'b',time,filty,'r');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,u,'r');
xlabel('time(s)');ylabel('u');
figure(3);
plot(time,D,'r');
xlabel('time(s)');ylabel('Disturbance signal');
```

仿真方法二

采用Simulink进行仿真。控制器采用积分分离PI控制，即当误差的绝对值小于等于0.80时，加入积分控制，仿真结果如图1-48和图1-49所示。

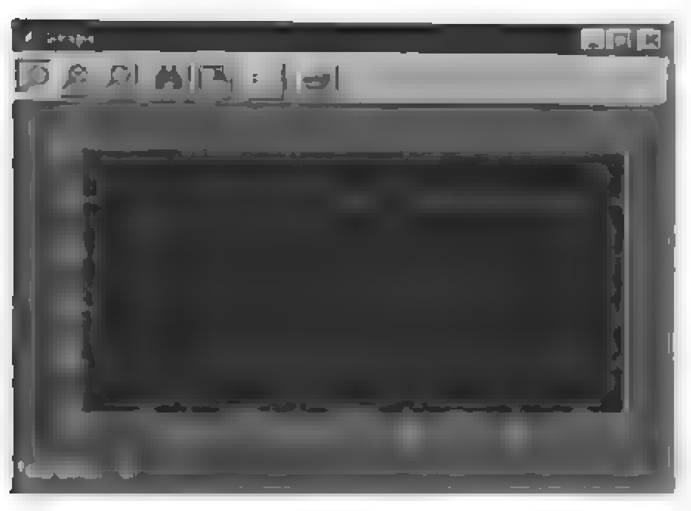


图 1-48　加入滤波器时 PID 控制阶跃响应

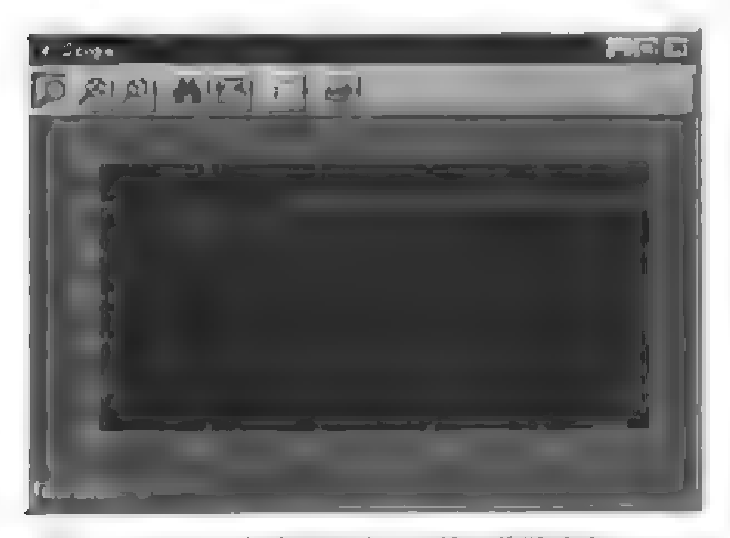


图 1-49　无滤波器时 PID 控制阶跃响应

仿真程序的初始化程序：chap1_19f.m。

Simulink 仿真程序：chap1_19.mdl，如图 1 50 和图 1-51 所示。

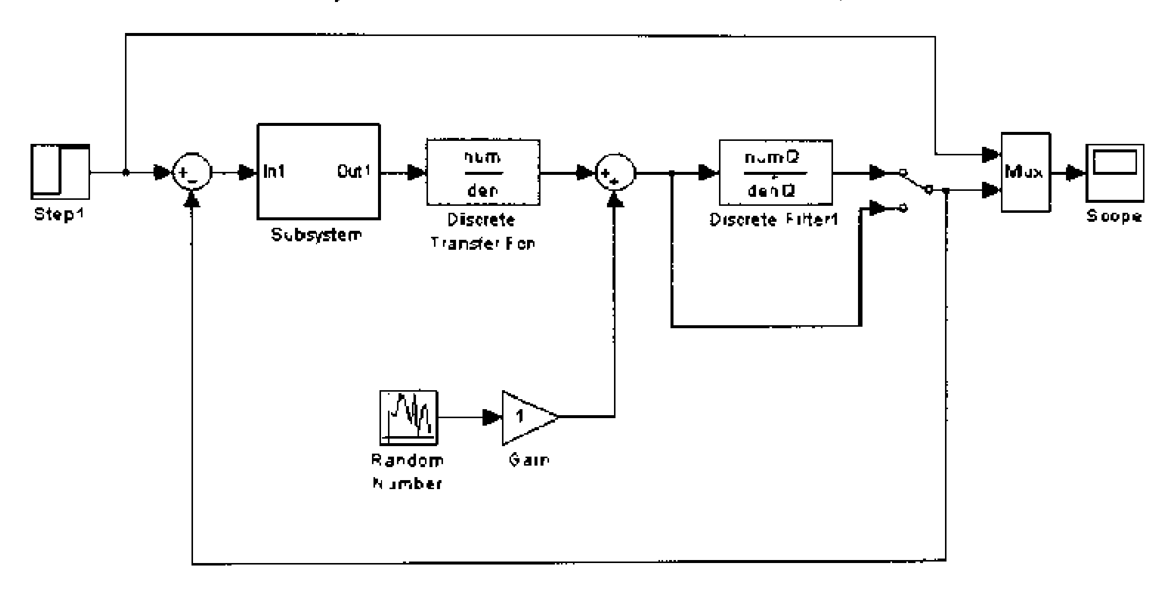


图 1-50　Simulink 仿真程序

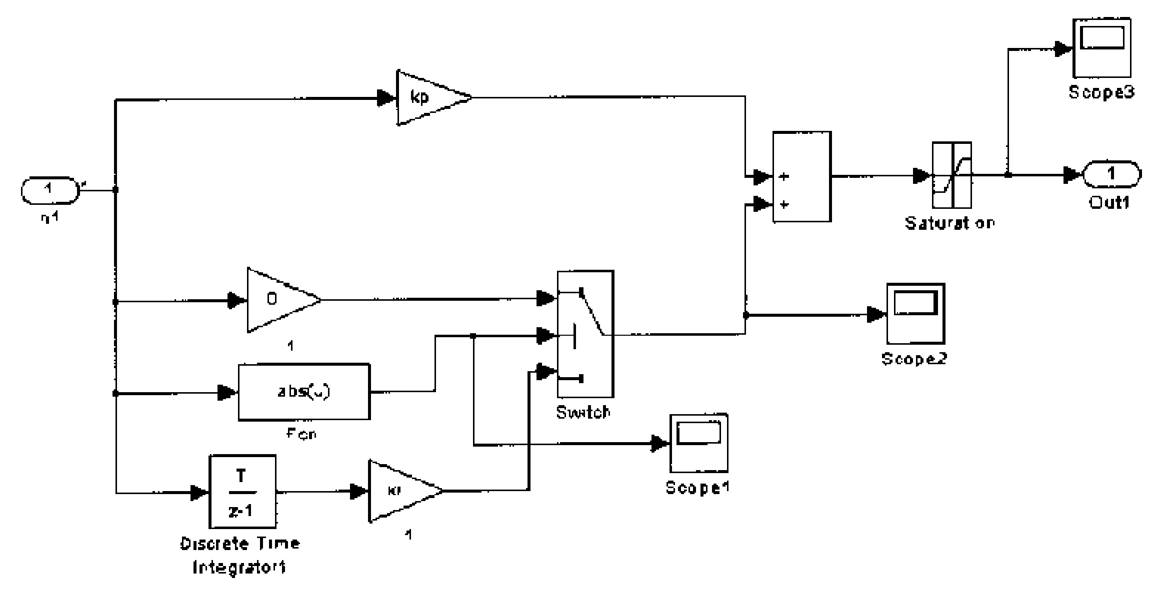


图 1 51　基于 Simulink 的 PI 控制器子程序

1.3.10　不完全微分 PID 控制算法及仿真

在 PID 控制中，微分信号的引入可改善系统的动态特性，但也易引进高频干扰，在误差扰动突变时尤其显出微分项的不足。若在控制算法中加入低通滤波器，则可使系统性能得到改善。

克服上述缺点的方法之一是，在 PID 算法中加入一个一阶惯性环节（低通滤波器）$G_f(s)=1/(1+T_f s)$，可使系统性能得到改善

不完全微分 PID 的结构如图 1-52（a）、（b）所示，其中图（a）是将低通滤波器直接加在微分环节上，图（b）是将低通滤波器加在整个 PID 控制器之后。下面以图（a）为例进行仿真说明不完全微分 PID 如何改进了普通 PID 的性能。

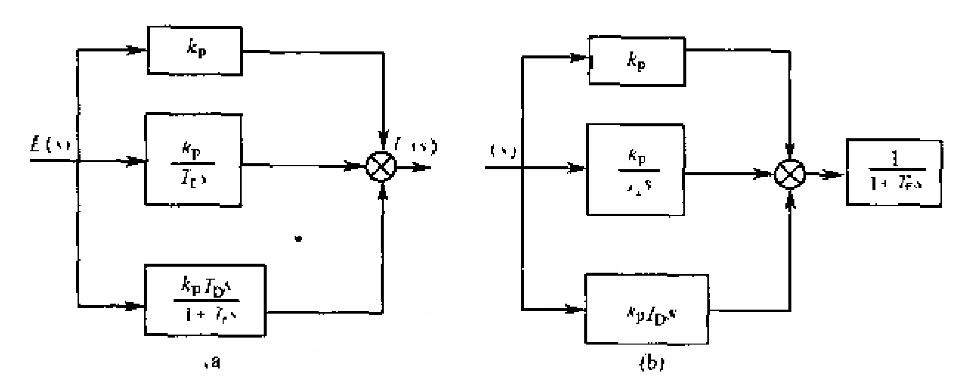


图 1-52　不完全微分算法结构图

对图（a）所示的不完全微分结构，其传递函数为：

$$U(s)=\left(k_p+\frac{k_p/T_I}{s}+\frac{k_pT_Ds}{T_fs+1}\right)E(s)=u_p(s)+u_I(s)+u_D(s) \tag{1.14}$$

将式（1.14）离散化为：

$$u(k)=u_p(k)+u_I(k)+u_D(k) \tag{1.15}$$

现将 $u_D(k)$ 推导：

$$u_D(s)=\frac{k_pT_Ds}{T_fs+1}E(s) \tag{1.16}$$

写成微分方程为：

$$u_D(k)+T_f\frac{du_D(t)}{dt}=k_pT_D\frac{derror(t)}{dt}$$

取采样时间为 T_s，将上式离散化为：

$$u_D(k)+T_f\frac{u_D(k)-u_D(k-1)}{T_s}=k_pT_D\frac{error(k)-error(k-1)}{T_s} \tag{1.17}$$

经整理得：

$$u_D(k)=\frac{T_f}{T_s+T_f}u_D(k-1)+k_p\frac{T_D}{T_s+T_f}(error(k)-error(k-1)) \tag{1.18}$$

令 $\alpha=\dfrac{T_f}{T_s+T_f}$，则 $\dfrac{T_s}{T_s+T_f}=1-\alpha$，显然有 $\alpha<1$，$1-\alpha<1$ 成立，则可得不完全微分算法：

$$u_D(k)=K_D(1-\alpha)(error(k)-error(k-1))+\alpha u_D(k-1) \tag{1.19}$$

式中，$K_D=k_p\cdot T_D/T_s$。

可见，不完全微分的 $u_D(k)$ 多了一项 $\alpha u_D(k-1)$，而原微分系数由 k_d 降至 $k_d(1-\alpha)$。

以上各式中，T_s 为采样时间，$T_s=\Delta t$，k_p 为比例系数，T_I 和 T_D 分别为积分时间常数和微分时间常数，T_f 为滤波器系数。

仿真实例

采用第一种不完全微分算法，被控对象为时滞系统传递函数：

$$G(s)=\frac{e^{-80s}}{60s+1}$$

在对象的输出端加幅值为 0.01 的随机信号。采样时间为 20ms。

低通滤波器为：

$$Q(s)=\frac{1}{180s+1}$$

取 $M=1$，采用具有不完全微分PID方法，其控制阶跃响应结果如图1-53所示。取 $M=2$，采用普通PID方法，阶跃响应结果如图1-54所示。由仿真结果可以看出，引入不完全微分后，能有效地克服普通PID的不足。尽管不完全微分PID控制算法比普通PID控制算法要复杂些，但由于其良好的控制特性，近年来得到越来越广泛的应用。

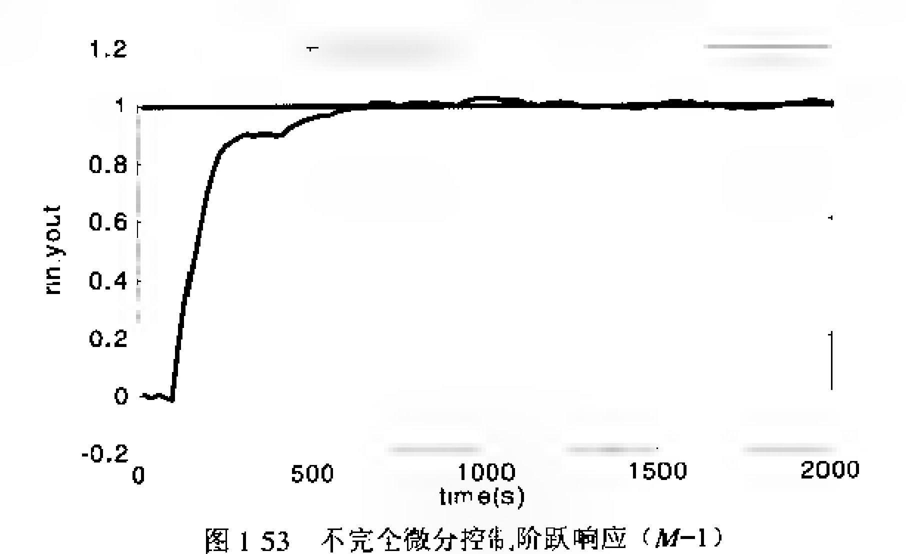


图1-53　不完全微分控制阶跃响应（$M=1$）

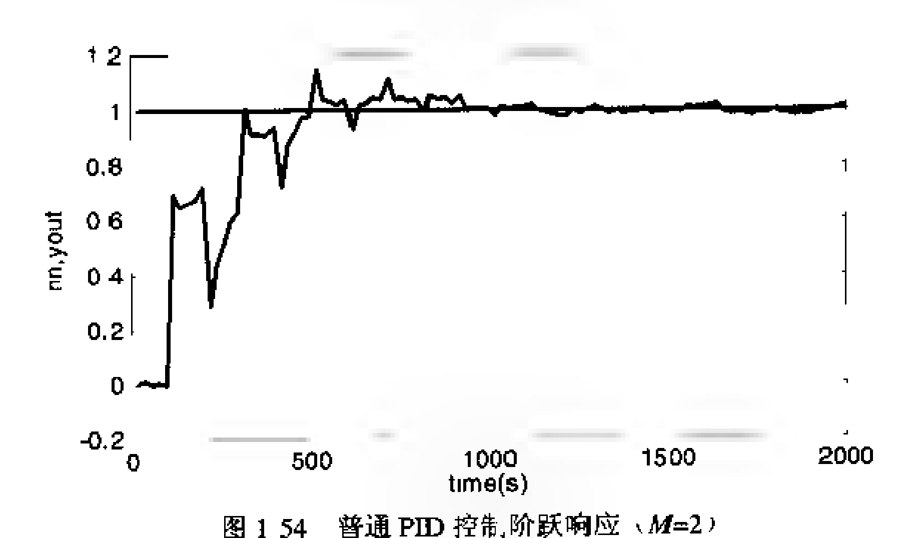


图1-54　普通PID控制阶跃响应（$M=2$）

仿真程序：chap1_20.m。

```
%PID Controller with Partial differential
clear all;
close all;

ts=20;
sys=tf([1],[60,1],'inputdelay',80);
dsys=c2d(sys,ts,'zoh');
```

```
[num,den]=tfdata(dsys,'v');

u_1=0;u_2=0;u_3=0;u_4=0;u_5=0;
ud_1=0;
y_1=0;y_2=0;y_3=0;
error_1=0;
ei=0;

for k=1:1:100
time(k)=k*ts;

rin(k)=1.0;

%Linear model
yout(k)=-den(2)*y_1+num(2)*u_5;

D(k)=0.01*rands(1);
yout(k)=yout(k)+D(k);

error(k)=rin(k)-yout(k);

%PID Controller with partly differential
ei=ei+error(k)*ts;
kc=0.30;
ki=0.0055;
TD=140;

kd=kc*TD/ts;

Tf=180;
Q=tf([1],[Tf,1]);   %Low Freq Signal Filter

M=2;
if M==1      %Using PID with Partial differential
    alfa=Tf/(ts+Tf);
    ud(k)=kd*(1-alfa)*(error(k)-error_1)+alfa*ud_1;
    u(k)=kc*error(k)+ud(k)+ki*ei;
   ud_1=ud(k);
elseif M==2  %Using Simple PID
    u(k)=kc*error(k)+kd*(error(k)-error_1)+ki*ei;
```

```
end

%Restricting the output of controller
if u(k)>=10
   u(k)=10;
end
if u(k)<=-10
   u(k)=-10;
end

u_5=u_4;u_4=u_3;u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);
error_1=error(k);
end
figure(1);
plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,u,'r');
xlabel('time(s)');ylabel('u');
figure(3);
plot(time,rin-yout,'r');
xlabel('time(s)');ylabel('error');
figure(4);
bode(Q,'r');
dcgain(Q);
```

1.3.11 微分先行 PID 控制算法及仿真

微分先行 PID 控制的结构如图 1-55 所示，其特点是只对输出量 yout(k) 进行微分，而对给定值 rin(k) 不进行微分。这样，在改变给定值时，输出不会改变，而被控量的变化通常是比较缓和的。这种输出量先行微分控制适用于给定值 rin(k) 频繁升降的场合，可以避免给定值升降时引起系统振荡，从而明显地改善了系统的动态特性。

令微分部分的传递函数为：

$$\frac{u_{\mathrm{D}}(s)}{y(s)}=\frac{T_{\mathrm{D}}s+1}{\gamma T_{\mathrm{D}}s+1} \qquad \gamma<1 \tag{1.20}$$

式中，$1/(\gamma T_{\mathrm{D}}s+1)$ 相当于低通滤波器。

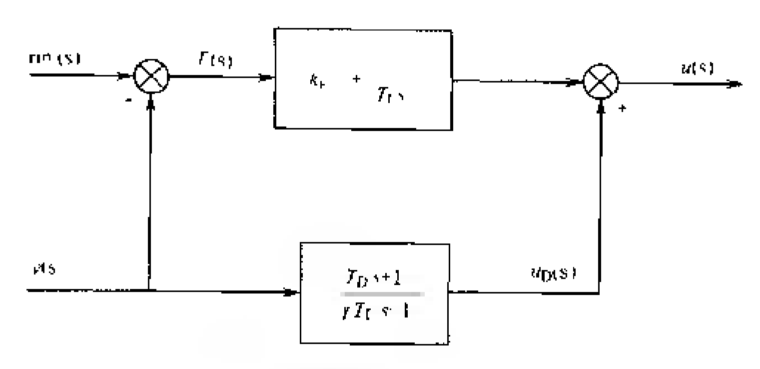


图 1-55　微分先行 PID 控制结构图

则有

$$\gamma T_{\mathrm{D}} \frac{\mathrm{d}u_{\mathrm{D}}}{\mathrm{d}t} + u_{\mathrm{D}} = T_{\mathrm{D}} \frac{\mathrm{d}y}{\mathrm{d}t} + y \tag{1.21}$$

由差分得：

$$\frac{\mathrm{d}u_{\mathrm{D}}}{\mathrm{d}t} \approx \frac{u_{\mathrm{D}}(k) - u_{\mathrm{D}}(k-1)}{T}$$

$$\frac{\mathrm{d}y}{\mathrm{d}t} \approx \frac{y(k) - y(k-1)}{T}$$

$$\gamma T_{\mathrm{D}} \frac{u_{\mathrm{D}}(k) - u_{\mathrm{D}}(k-1)}{T} + u_{\mathrm{D}}(k) = T_{\mathrm{D}} \frac{y(k) - y(k-1)}{T} + y(k)$$

$$u_{\mathrm{D}}(k) = \left(\frac{\gamma T_{\mathrm{D}}}{\gamma T_{\mathrm{D}} + T}\right) u_{\mathrm{D}}(k-1) + \left(\frac{T_{\mathrm{D}} + T}{\gamma T_{\mathrm{D}} + T}\right) y(k) - \left(\frac{T_{\mathrm{D}}}{\gamma T_{\mathrm{D}} + T}\right) y(k-1)$$

$$u_{\mathrm{D}}(k) = c_1 u_{\mathrm{D}}(k-1) + c_2 y(k) - c_3 y(k-1) \tag{1.22}$$

其中，

$$c_1 = \frac{\gamma T_{\mathrm{D}}}{\gamma T_{\mathrm{D}} + T}, \quad c_2 = \frac{T_{\mathrm{D}} + T}{\gamma T_{\mathrm{D}} + T}, \quad c_3 = \frac{T_{\mathrm{D}}}{\gamma T_{\mathrm{D}} + T} \tag{1.23}$$

PID 控制部分传递函数为：

$$\frac{u_{\mathrm{pI}}(s)}{E(s)} = k_{\mathrm{p}}\left(1 + \frac{1}{T_{\mathrm{I}} s}\right) \tag{1.24}$$

式中，T_{I} 为积分时间常数。

离散控制律为：

$$u(k) = u_{\mathrm{pI}}(k) + u_{\mathrm{D}}(k) \tag{1.25}$$

仿真实例

设被控对象为一个延迟对象：

$$G(s) = \frac{\mathrm{e}^{-80s}}{60s + 1}$$

采样时间为 20s，延迟时间为 4 个采样时间，即 80s。采用 PID 控制器进行阶跃响应。输入信号为带有高频干扰的方波信号：$\mathrm{rin}(t) = 1.0\mathrm{sgn}(\sin(0.0005\pi t) + 0.05\sin(0.03\pi t))$。

取 $M = 1$，采用微分先行 PID 控制方法，其方波响应结果如图 1-56 和图 1-57 所示。取 $M = 2$，采用普通 PID 方法，其方波响应控制结果如图 1-58 和图 1-59 所示。由仿真结果可以看出，对于给定值 rin(k) 频繁升降的场合，引入微分先行后，可以避免给定值升降时所引

起的系统振荡，明显地改善了系统的动态特性。

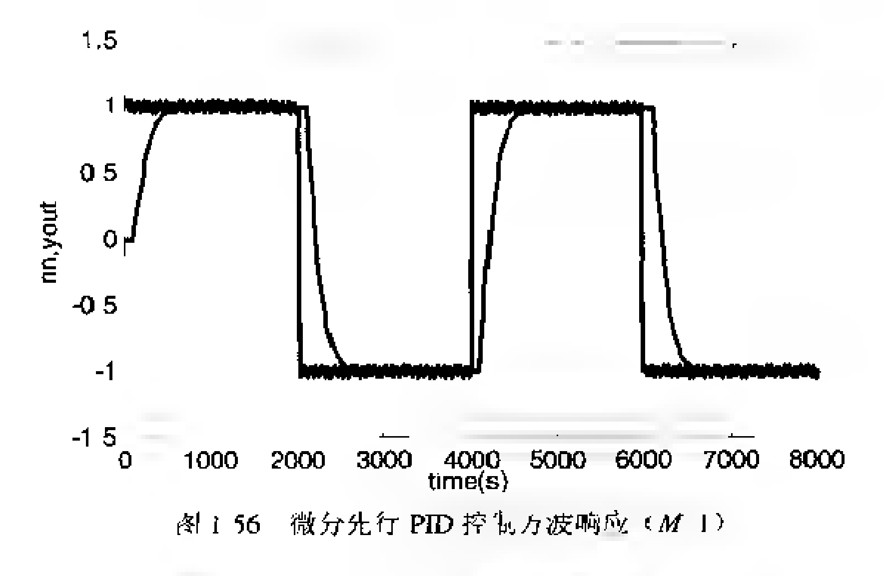


图1-56 微分先行PID控制方波响应（M=1）

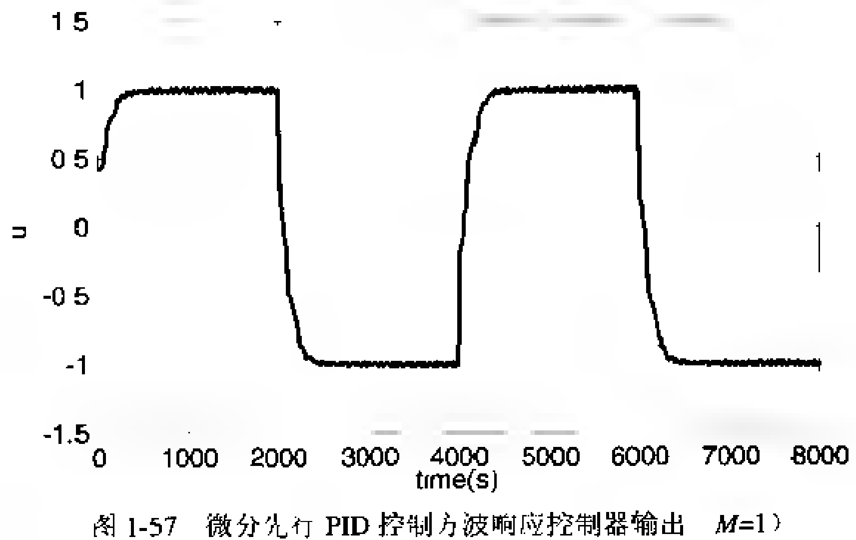


图1-57 微分先行PID控制方波响应控制器输出（M=1）

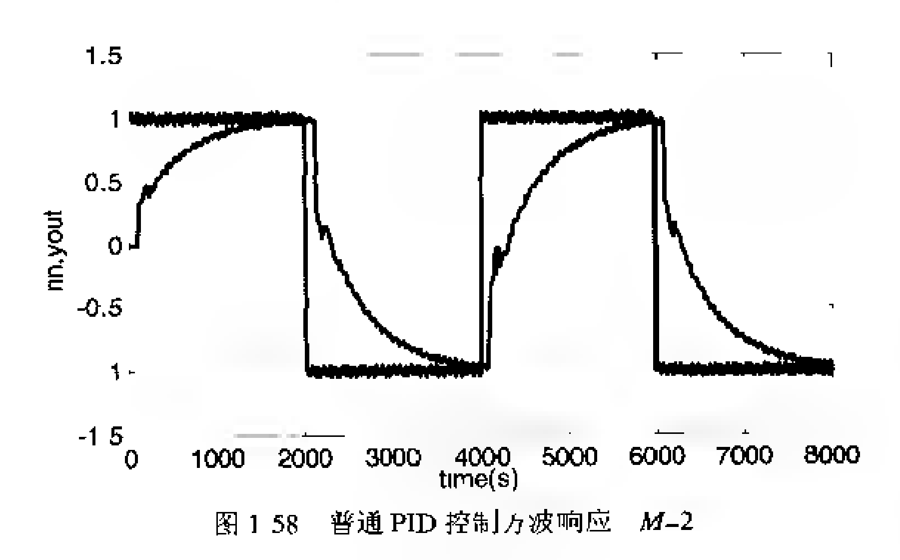


图1-58 普通PID控制方波响应（M=2）

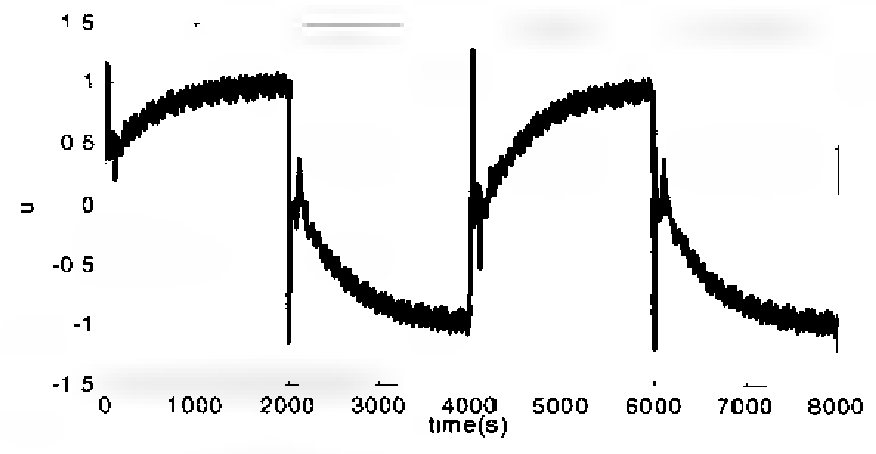


图 1-59　普通 PID 控制方波响应控制器输出（M=2）

仿真程序：chap1_21.m。

```
%PID Controller with differential in advance
clear all;
close all;

ts=20;
sys=tf([1],[60,1],'inputdelay',80);
dsys=c2d(sys,ts,'zoh');
[num,den]=tfdata(dsys,'v');

u_1=0;u_2=0;u_3=0;u_4=0;u_5=0;
ud_1=0;
y_1=0;y_2=0;y_3=0;
error_1=0;error_2=0;
ei=0;
for k=1:1:400
time(k)=k*ts;

%Linear model
yout(k)=-den(2)*y_1+num(2)*u_5;

kp=0.36;kd=14;ki=0.0021;

rin(k)=1.0*sign(sin(0.00025*2*pi*k*ts));
rin(k)=rin(k)+0.05*sin(0.03*pi*k*ts);

error(k)=rin(k)-yout(k);
ei=ei+error(k)*ts;

gama=0.50;
Td=kd/kp;
Ti=0.5;
```

```
c1=gama*Td/(gama*Td+ts);
c2=(Td+ts)/(gama*Td+ts);
c3=Td/(gama*Td+ts);

M=2;
if M==1      %PID Control with differential in advance
    ud(k)=c1*ud_1+c2*yout(k)-c3*y_1;
    u(k)=kp*error(k)+ud(k)+ki*ei;
elseif M==2  %Simple PID Control
    u(k)=kp*error(k)+kd*(error(k)-error_1)/ts+ki*ei;
end

if u(k)>=110
   u(k)=110;
end
if u(k)<=-110
   u(k)=-110;
end
%Update parameters
u_5=u_4;u_4=u_3;u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);

error_2=error_1;
error_1=error(k);
end
figure(1);
plot(time,rin,'r',time,yout,'b');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,u,'r');
xlabel('time(s)');ylabel('u');
```

1.3.12 带死区的PID控制算法及仿真

在计算机控制系统中，某些系统为了避免控制作用过于频繁，消除由于频繁动作所引起的振荡，可采用带死区的PID控制算法，控制算式为：

$$e(k)=\begin{cases}0 & |e(k)|<|e_0| \\ e(k) & |e(k)|>|e_0|\end{cases} \tag{1.26}$$

式中，$e(k)$为位置跟踪偏差，e_0是一个可调参数，其具体数值可根据实际控制对象由实验确定。若e_0值太小，会使控制动作过于频繁，达不到稳定被控对象的目的；若e_0太大，则系统将产生较大的滞后。

带死区的控制系统实际上是一个非线性系统，当$|e(k)|<|e_0|$时，数字调节器输出为零；

当$|e(k)|>|e_0|$时，数字输出调节器有 PID 输出。带死区的 PID 控制算法流程图如图 1-60 所示。

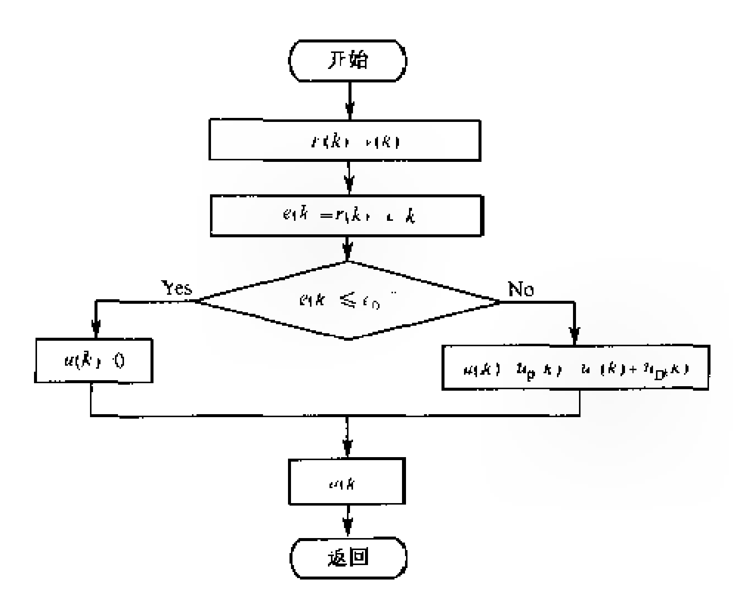


图 1-60 带死区的 PID 控制算法程序框图

仿真实例

设被控制对象为：

$$G(s)=\frac{523500}{s^3+87.35s^2+10470s}$$

采样时间为 1ms，对象输出上有一个幅值为 0.5 的正态分布的随机干扰信号。采用积分分离式 PID 控制算法进行阶跃响应，取$\varepsilon=0.20$，死区参数e_0=0.10，采用低通滤波器对对象输出信号进行滤波，滤波器为：

$$Q(s)=\frac{1}{0.04s+1}$$

取$M=1$，采用一般积分分离式 PID 控制方法，其控制结果如图 1-61 所示。取$M=2$，采用带死区的积分分离式 PID 控制方法，其控制结果如图 1-62 所示。由仿真结果可以看出，引入带死区 PID 控制后，控制器输出更加平稳。

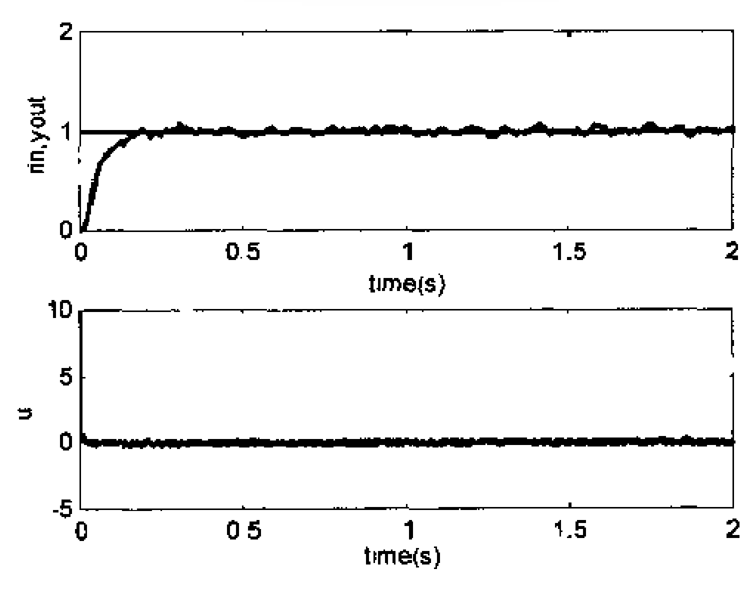


图 1-61 不带死区 PID 控制（M=1）

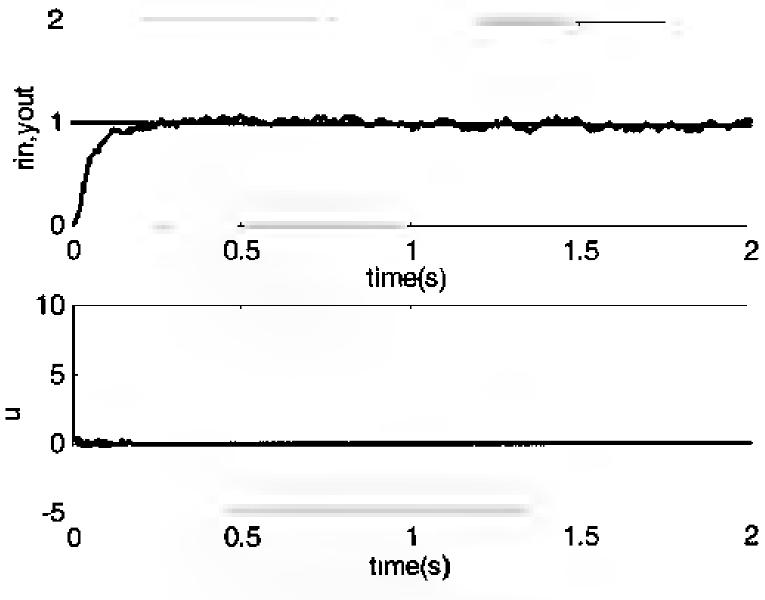


图 1-62　带死区 PID 控制（M=2）

仿真程序：chap1_22.m。

```
%PID Controller with dead zone
clear all;
close all;

ts=0.001;
sys=tf(5.235e005,[1,87.35,1.047e004,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

u_1=0;u_2=0;u_3=0;u_4=0;u_5=0;
y_1=0;y_2=0;y_3=0;
yy_1=0;
error_1=0;error_2=0;ei=0;

sys1=tf([1],[0.04,1]);   %Low Freq Signal Filter
dsys1=c2d(sys1,ts,'tustin');
[num1,den1]=tfdata(dsys1,'v');
f_1=0;

for k=1:1:2000
time(k)=k*ts;

rin(k)=1; %Step Signal

%Linear model
yout(k)=-den(2)*y_1-den(3)*y_2-den(4)*y_3+num(2)*u_1+...
      num(3)*u_2+num(4)*u_3;
```

```
D(k)=0.50*rands(1);      %Disturbance signal
yyout(k)=yout(k)+D(k);

%Low frequency filter
filty(k)=-den1(2)*f_1+num1(1)*(yyout(k)+yy_1);
error(k)=rin(k)-filty(k);

if abs(error(k))<=0.20
   ei=ei+error(k)*ts;
else
   ei=0;
end

kp=0.50;ki=0.10;kd=0.020;
u(k)=kp*error(k)+ki*ei+kd*(error(k)-error_1)/ts;

M=2;
if M==1
    u(k)=u(k);
elseif M==2  %Using Dead zone
   if abs(error(k))<=0.10
      u(k)=0;
   end
end

if u(k)>=10
   u(k)=10;
end
if u(k)<=-10
   u(k)=-10;
end
%--------------Return of PID parameters--------------
rin_1=rin(k);
u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);

f_1=filty(k);
yy_1=yyout(k);

error_2=error_1;
error_1=error(k);
end
figure(1);
subplot(211);
```

```
plot(time,rin,'r',time,filty,'b');
xlabel('time(s)');ylabel('rin,yout');
subplot(212);
plot(time,u,'r');
xlabel('time(s)');ylabel('u');
figure(2);
plot(time,D,'r');
xlabel('time(s)');ylabel('Disturbance signal');
```

1.3.13 基于前馈补偿的 PID 控制算法及仿真

在高精度伺服控制中，前馈控制可用来提高系统的跟踪性能。经典控制理论中的前馈控制设计是基于复合控制思想，当闭环系统为连续系统时，使前馈环节与闭环系统的传递函数之积为 1，从而实现输出完全复现输入。作者利用前馈控制的思想，针对 PID 控制设计了前馈补偿，以提高系统的跟踪性能，其结构如图 1-63 所示。

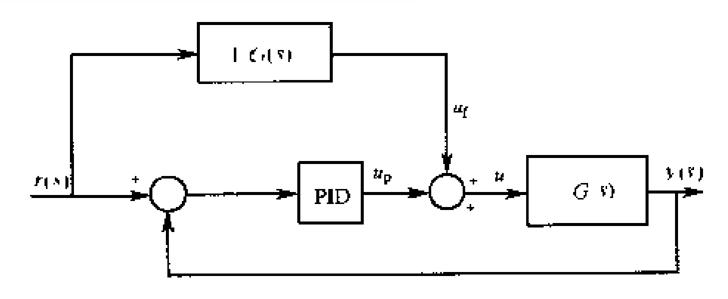


图 1-63　PID 前馈控制结构

设计前馈补偿控制器为：

$$u_f(s) = r(s)\frac{1}{G(s)} \tag{1.27}$$

总控制输出为 PID 控制输出加前馈控制输出：

$$u(t) = u_p(t) + u_f(t) \tag{1.28}$$

写成离散形式为：

$$u(k) = u_p(k) + u_f(k) \tag{1.29}$$

仿真实例

设被控制对象为：

$$G(s) = \frac{133}{s^2 + 25s}$$

输入信号为：$r(k) = 0.5\sin(6\pi t)$，采样时间为 1ms。

$$u_f(t) = \frac{25}{133}\dot{r}(t) + \frac{1}{133}\ddot{r}(t)$$

写成离散形式为：

$$u_f(k) = \frac{25}{133}\dot{r}(k) + \frac{1}{133}\ddot{r}(k)$$

只采用 PID 正弦跟踪控制方法的结果如图 1-64 和图 1-65 所示，采用前馈 PID 控制方法

的跟踪结果如图 1-66 和图 1-67 所示。可见通过前馈补偿可大大提高系统的跟踪性能。

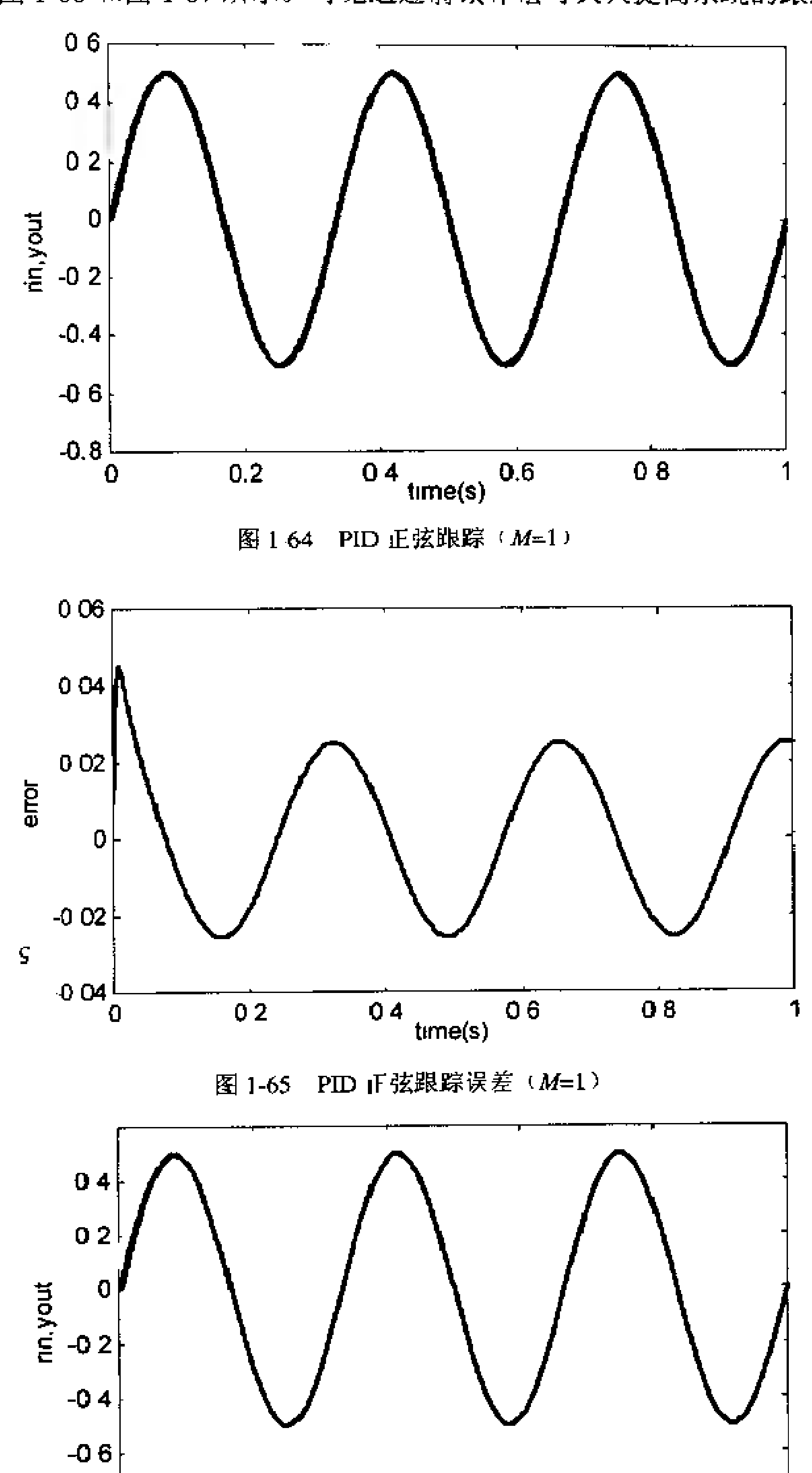


图 1-64 PID 正弦跟踪（M=1）

图 1-65 PID 正弦跟踪误差（M=1）

图 1-66 PID 加前馈补偿正弦跟踪（M=2）

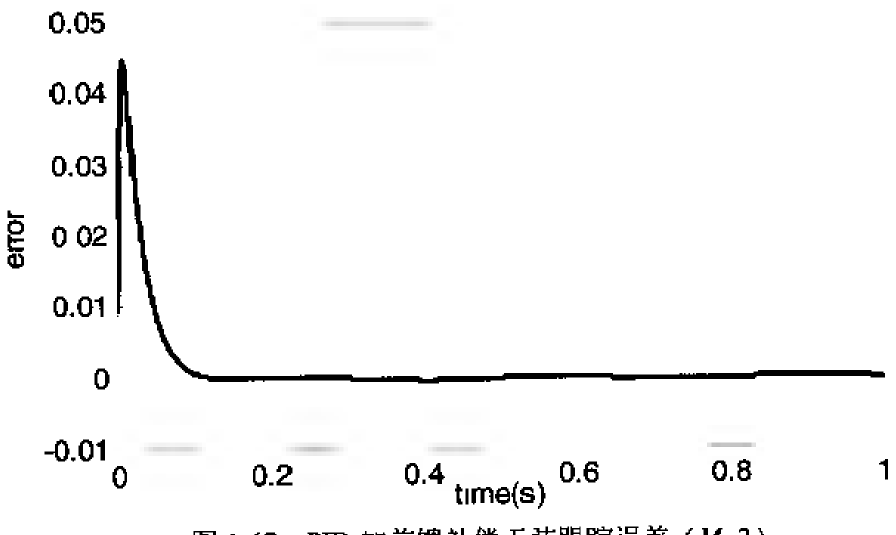


图 1-67　PID 加前馈补偿正弦跟踪误差（M=2）

仿真程序：chap1_23.m。

```
%PID Feedforward Controller
clear all;
close all;

ts=0.001;
sys=tf(133,[1,25,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

u_1=0;u_2=0;
y_1=0;y_2=0;

error_1=0;ei=0;
for k=1:1:1000
time(k)=k*ts;

A=0.5;F=3.0;
rin(k)=A*sin(F*2*pi*k*ts);
drin(k)=A*F*2*pi*cos(F*2*pi*k*ts);
ddrin(k)=-A*F*2*pi*F*2*pi*sin(F*2*pi*k*ts);

%Linear model
yout(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;

error(k)=rin(k)-yout(k);

ei=ei+error(k)*ts;
```

```
up(k)=80*error(k)+20*ei+2.0*(error(k)-error_1)/ts;

uf(k)=25/133*drin(k)+1/133*ddrin(k);

M=2;
if M==1         %Only using PID
    u(k)=up(k);
elseif M==2     %PID+Feedforward
    u(k)=up(k)+uf(k);
end

if u(k)>=10
   u(k)=10;
end
if u(k)<=-10
   u(k)=-10;
end

u_2=u_1;u_1=u(k);
y_2=y_1;y_1=yout(k);
error_1=error(k);
end
figure(1);
plot(time,rin,'r',time,yout,'b');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,error,'r');
xlabel('time(s)');ylabel('error');
figure(3);
plot(time,up,'k',time,uf,'b',time,u,'r');
xlabel('time(s)');ylabel('up,uf,u');
```

1.3.14 步进式 PID 控制算法及仿真

在较大阶跃响应时，很容易产生超调。采用步进式积分分离 PID 控制，该方法不直接对阶跃信号进行响应，而是使输入指令信号一步一步地逼近所要求的阶跃信号，可使对象运行平稳，适用于高精度伺服系统的位置跟踪。

仿真实例

设被控制对象为：

$$G(s)=\frac{523500}{s^3+87.35s^2+10470s}$$

采样时间为 1ms，输入指令信号为 $\mathrm{rd}=20$。采用本控制算法进行阶跃响应。其中 $M=1$

时为积分分离式PID控制，响应结果如图1-68所示，$M=2$时为步进式积分分离PID控制，响应结果及输入信号的变化如图1-69和图1-70所示。

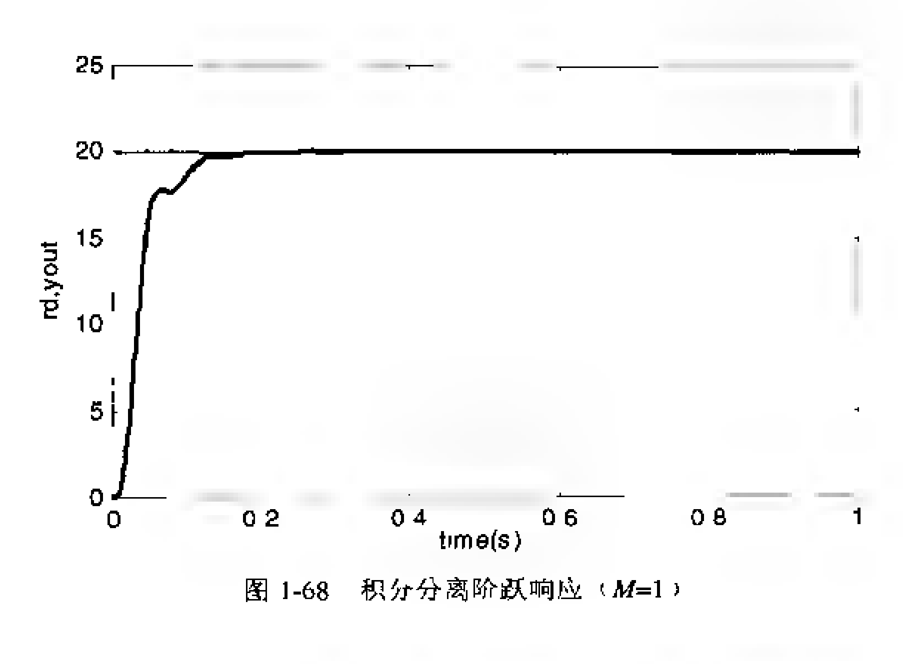


图1-68 积分分离阶跃响应（$M=1$）

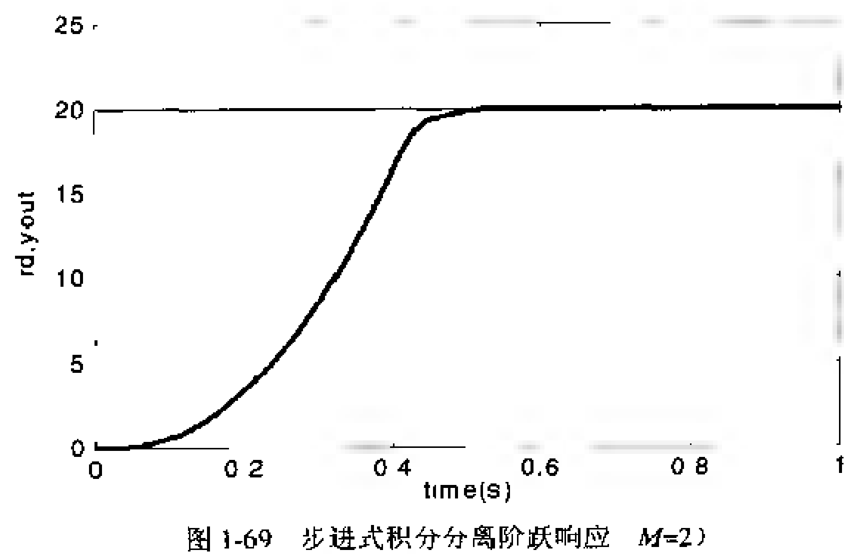


图1-69 步进式积分分离阶跃响应（$M=2$）

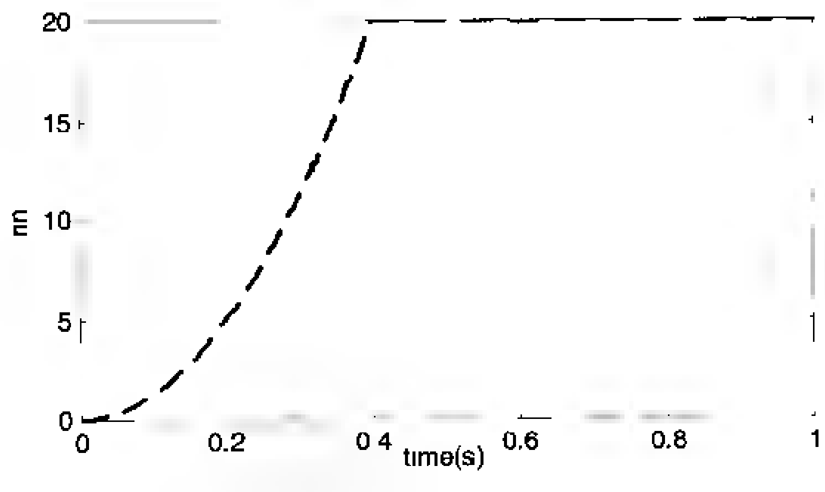


图1-70 步进式阶跃信号rn的变化（$M=2$）

在步进式 PID 控制的仿真中，实际输入指令 rin(k) 采用 0.25 的步长变化，逐渐逼近输入指令信号 rd。仿真结果表明，采用积分分离式 PID 控制，响应速度快，但阶跃响应不平稳；而采用步进式 PID 控制，虽然响应速度慢，但阶跃响应平稳，具有很好的工程实用价值。

仿真程序：chap1_24.m。

```
%PID Control with Gradual approaching input value
clear all;
close all;

ts=0.001;
sys=tf(5.235e005,[1,87.35,1.047e004,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

u_1=0;u_2=0;u_3=0;u_4=0;u_5=0;
y_1=0;y_2=0;y_3=0;
error_1=0;error_2=0;ei=0;

kp=0.50;ki=0.05;
rate=0.25;
rini=0.0;

for k=1:1:1000
time(k)=k*ts;
rd=20;   %Step Signal

%Linear model
yout(k)=-den(2)*y_1-den(3)*y_2-den(4)*y_3+num(2)*u_1+num(3)*u_2+num(4)*u_3;

M=2;
if M==1   %Using simple PID
   rin(k)=rd;
   error(k)=rin(k)-yout(k);
end
if M==2   %Using Gradual approaching input value
if rini<rd-0.25
   rini=rini+k*ts*rate;
elseif rini>rd+0.25
   rini=rini-k*ts*rate;
else
   rini=rd;
end
```

```
   rin(k)=rin1;
   error(k)=rin(k)-yout(k);
end

%PID with I separation
if abs(error(k))<=0.8
   ei=ei+error(k)*ts;
else
   ei=0;
end
u(k)=kp*error(k)+ki*ei;

if u(k)>=10
   u(k)=10;
end
if u(k)<=-10
   u(k)=-10;
end
%--------  Return of PID parameters   --------
rin_1=rin(k);
u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);

error_2=error_1;
error_1=error(k);
end
figure(1);
plot(time,rd,'b',time,yout,'r');
xlabel('time(s)');ylabel('rd,yout');
figure(2);
plot(time,u,'r');
xlabel('time(s)');ylabel('u');
figure(3);
plot(time,rin,'r.');
xlabel('time(s)');ylabel('rin');
```

1.3.15 PID 控制的方波响应

设被控对象为一个延迟对象：

$$G(s)=\frac{e^{-80s}}{60s+1}$$

采样时间为 20s，延迟时间为 4 个采样时间，即 80s，被控对象离散化为：

$$y(k) = -\text{den}(2)y(k-1) + \text{num}(2)u(k-5)$$

由于方波信号的速度、加速度不连续，当位置跟踪指令为方波信号时，如采用滤波器对指令信号进行滤波，将滤波输出作为给定信号，可使方波响应及执行器的动作更加平稳，在工程上具有一定意义。

二阶离散滤波器的设计原理为：

$$F(z^{-1}) = a_1 + a_2 z^{-1} + a z^{-2}$$
$$2a_1 + a_2 = 1$$

取方波信号为 $\text{rin}(t) = \text{sgn}(0.0001\pi t)$，滤波器参数取 $a_1 = 0.10, a_2 = 0.80$。

分两种情况进行仿真：当 $M = 1$ 时，为普通方波指令信号，方波响应结果如图 1-71 和图 1-72 所示；当 $M = 2$ 时，为加滤波方波指令信号，方波响应结果如图 1-73 和图 1-74 所示。

可见，将方波指令信号加滤波后，方波响应更加平稳，控制输入信号的抖动消除。

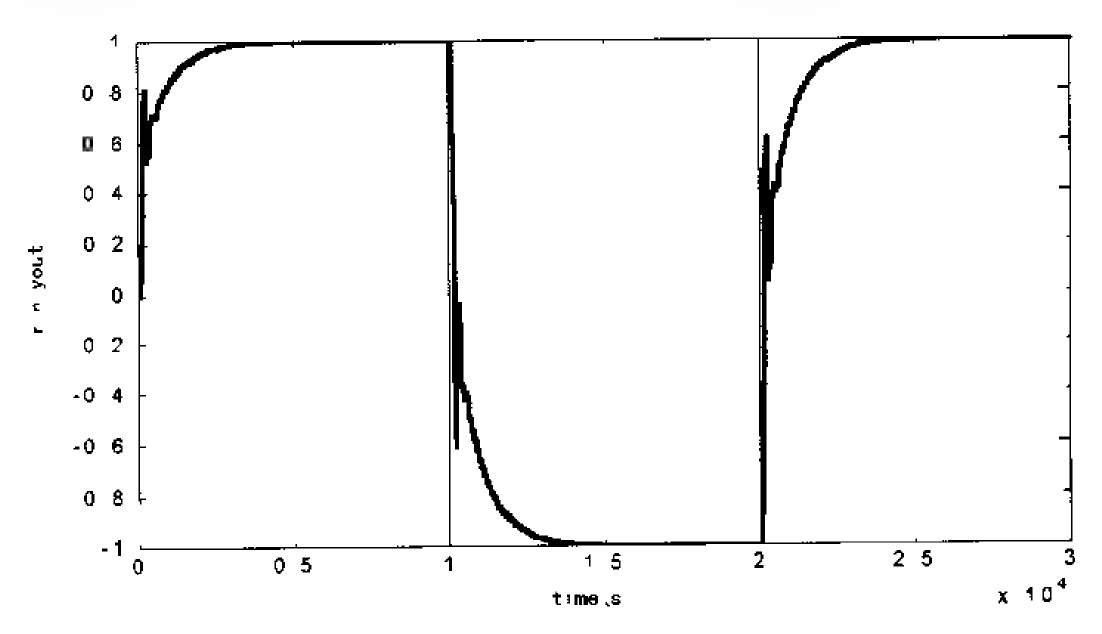


图 1-71　普通方波指令信号的 PID 响应（M=1）

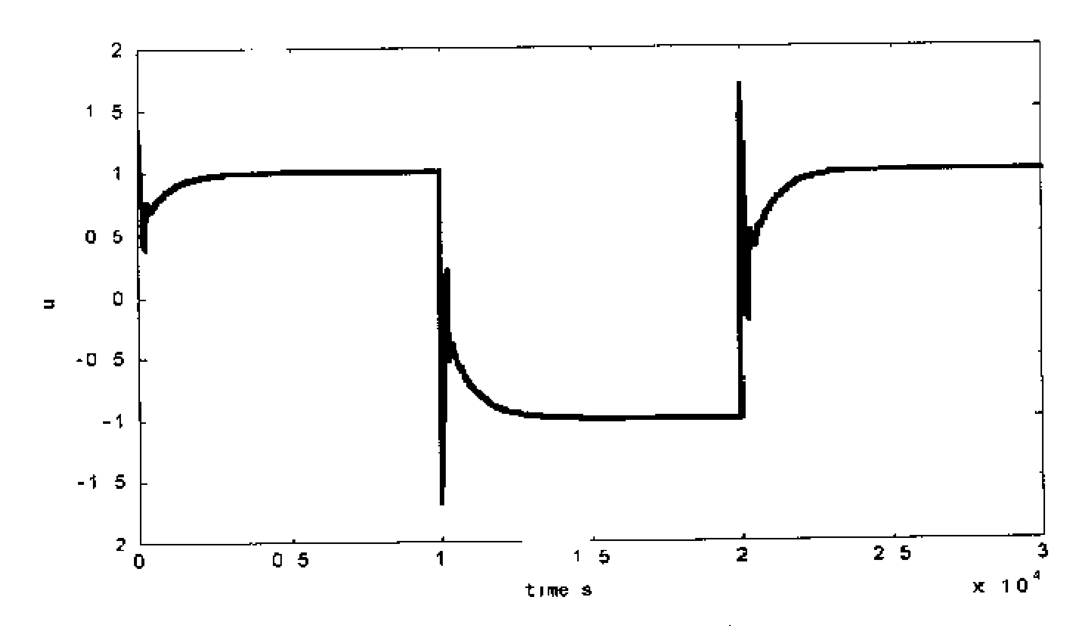


图 1-72　普通方波指令信号的 PID 控制输入（M=1）

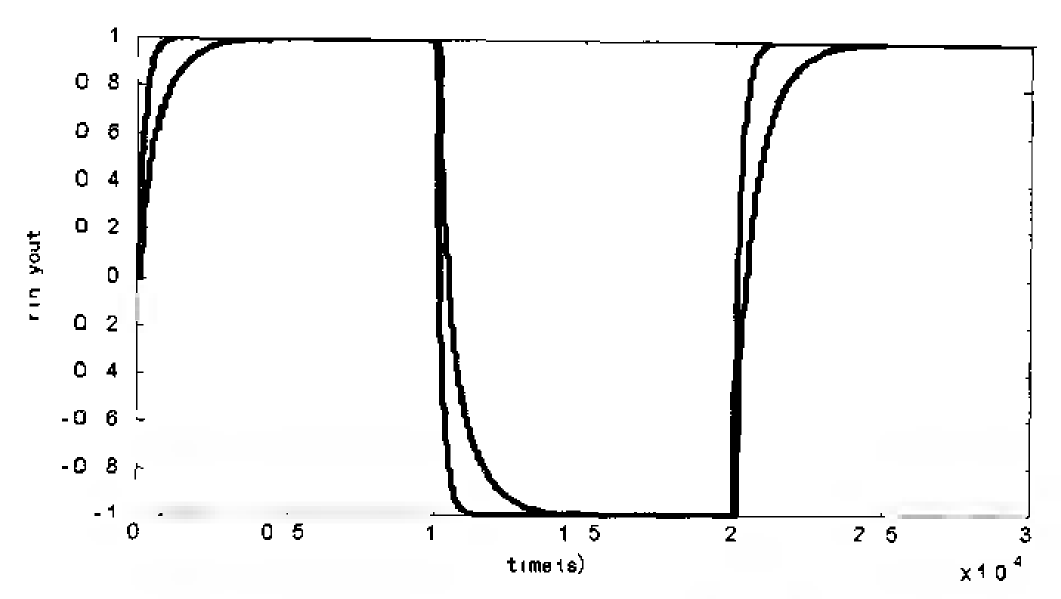


图 1-73　带滤波器的方波指令信号 PID 响应　M=2

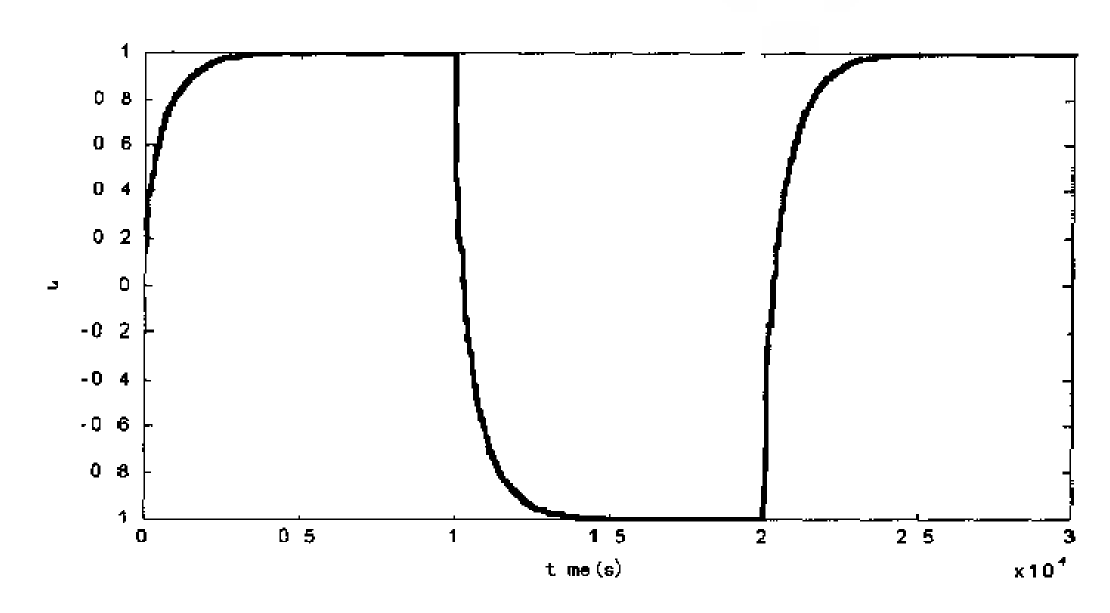


图 1-74　带滤波器的方波指令信号 PID 控制输入　M=2）

仿真程序：chap1_25.m。

```
%PID Controller for Square Tracking with Filtered Signal
clear all;
close all;

ts=20;
sys=tf([1],[60,1],'inputdelay',80);
dsys=c2d(sys,ts,'zoh');
[num,den]=tfdata(dsys,'v');

u_1=0;u_2=0;u_3=0;u_4=0;u_5=0;
```

```
y_1=0;
error_1=0;
ei=0;
rin_1=0;rin_2=0;
for k=1:1:500
time(k)=k*ts;

rin(k)=1.0*sign(sin(0.00005*2*pi*k*ts));

M=1;
switch M;
case 1
   rin(k)=rin(k);
case 2
   rin(k)=0.10*rin(k)+0.80*rin_1+0.10*rin_2;
end

%Linear model
yout(k)=-den(2)*y_1+num(2)*u_5;

kp=0.80;
kd=10;
ki=0.002;

error(k)=rin(k)-yout(k);
ei=ei+error(k)*ts;

u(k)=kp*error(k)+kd*(error(k)-error_1)/ts+ki*ei;

%Update parameters
u_5=u_4;u_4=u_3;u_3=u_2;u_2=u_1;u_1=u(k);
y_1=yout(k);

error_2=error_1;
error_1=error(k);
rin_2=rin_1;
rin_1=rin(k);
end
```

```
figure(1);
plot(time,rin,'k',time,yout,'k');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,u,'k');
xlabel('time(s)');ylabel('u');
```

1.3.16　一种离散微分-跟踪器

韩京清教授提出的一种微分-跟踪器[5]，其离散形式为：

$$
\begin{cases}
x_1(k+1) = x_1(k) + Tx_2(k) \\
x_2(k+1) = x_2(k) + T\mathrm{fst}(x_1(k), x_2(k), u(k), r, h)
\end{cases}
$$

式中，T 为采样周期，$u(k)$ 为第 k 时刻的输入信号，r 为决定跟踪快慢的参数，h 为输入信号被噪声污染时，决定滤波效果的参数。

fst 函数描述如下：

$$
\delta = rh,\quad \delta_0 = \delta h,\quad y = x_1 - u + hx_2,\quad a_0 = \sqrt{\delta^2 + 8r|y|}
$$

$$
a = \begin{cases}
x_2 + y/h & |y| \leqslant \delta_0 \\
x_2 + 0.5(a_0 - \delta)\mathrm{sign}(y) & |y| > \delta_0
\end{cases}
$$

$$
\mathrm{fst} = \begin{cases}
-ra/\delta & |a| \leqslant \delta \\
r\mathrm{sign}(a) & |a| > \delta
\end{cases}
$$

仿真实例一

验证离散微分-跟踪器的滤波性能。

在离散微分-跟踪器中，取 $r = 1800$，$h = 0.015$。采样时间为 1ms，输入信号为幅值为 1.0，频率为 1.0Hz 的正弦信号，在该信号上带有幅值为 0.10 的随机高频干扰信号。采用离散微分-跟踪器滤掉噪声信号。仿真结果如图 1-75 和图 1-76 所示。仿真结果表明，该滤波器对随机信号具有很好的滤波作用。

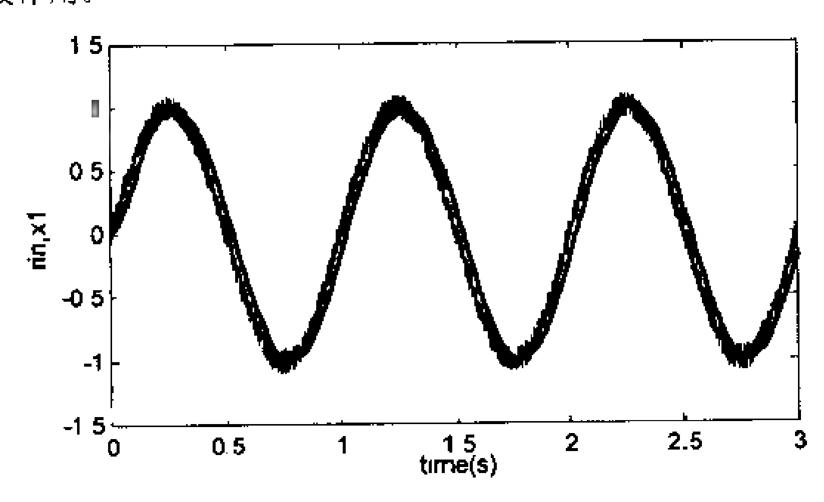


图 1-75　带噪声信号及滤波后信号

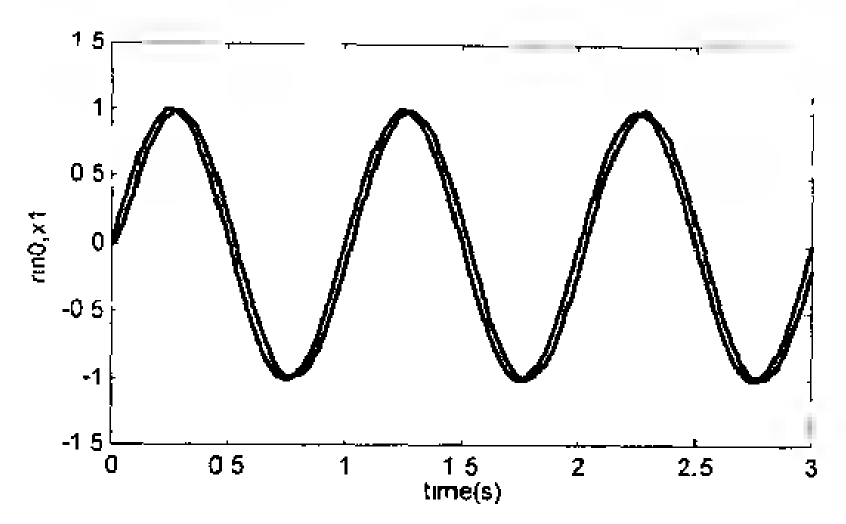


图 1-76　理想信号及滤波后信号

仿真程序：chap1_26.m。

```
%Filter with tracker and differentiation
clear all;
close all;

ts=0.001;
x=[0,0];
for k=1:1:3000
time(k)=k*ts;

u0(k)=1.0*sin(1*2*pi*k*ts);
u(k)=u0(k)+0.1*rands(1);

r=1800;
h=0.015;
T=ts;

delta=r*h;
delta0=delta*h;
y=x(1)-u(k)+h*x(2);
a0=sqrt(delta*delta+8*r*abs(y));
if abs(y)<=delta0
   a=x(2)+y/h;
else
   a=x(2)+0.5*(a0-delta)*sign(y);
end
if abs(a)<=delta
   fst2=-r*a/delta;
```

```
else
    fst2=r*sign(a);
 end

x(1)=x(1)+T*x(2);
x(2)=x(2)+T*fst2;
rin0(k)=u0;
rin(k)=u;
x1(k)=x(1);
end
figure(1);
plot(time,rin,'k',time,x1,'k');
xlabel('time(s)');ylabel('rin,x1');
figure(2);
plot(time,rin0,'k',time,x1,'k');
xlabel('time(s)');ylabel('rin0,x1');
```

仿真实例二

采用一种离散微分-跟踪器的PID控制。

设被控制对象为三阶传递函数：

$$G_p(s)=\frac{523500}{s^3+87.35s^2+10470s}$$

采样时间为1ms，指令信号$rin(k)=20$，干扰信号为0.5rands(1)，加在对象的输出端。在离散微分-跟踪器中，取$r=2000$，$h=0.02$。采用积分分离PI控制，取$k_p=0.12$，$k_i=0.015$。采用M语言进行仿真。分两种情况进行：当$M=1$时，为加干扰信号未加滤波；当$M=2$时，为加干扰信号加滤波。阶跃响应结果如图1-77和图1-78所示。

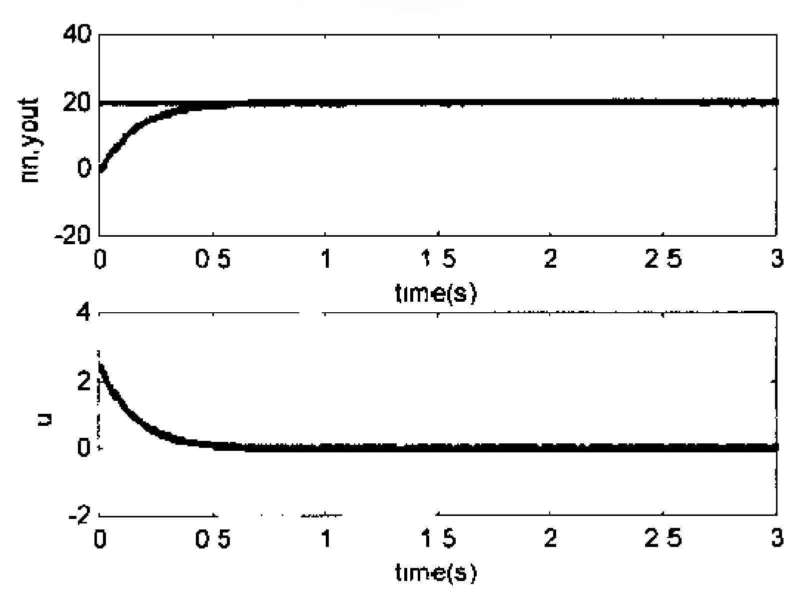


图1-77　无滤波器时PID控制阶跃响应及控制输入（M=1）

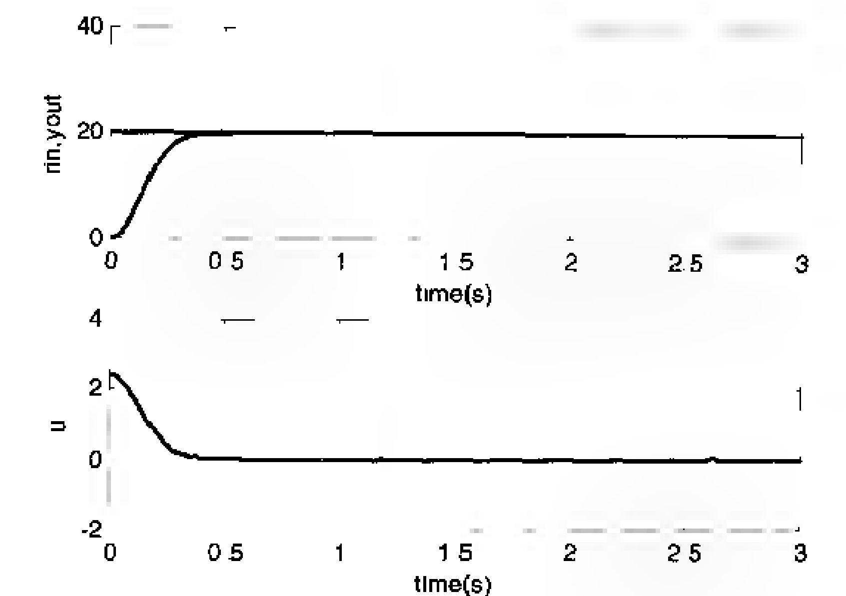


图 1.78　加入滤波器后 PID 控制阶跃响应及控制输入（$M=2$）

仿真程序：chap1_27.m。

```
%PID Controller with Partial differential
clear all;
close all;

ts=0.001;
sys=tf(5.235e005,[1,87.35,1.047e004,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

u_1=0;u_2=0;u_3=0;u_4=0;u_5=0;
y_1=0;y_2=0;y_3=0;

kp=0.12;ki=0.015;
x=[0,0];
for k=1:1:3000
time(k)=k*ts;
rin(k)=20;  %Step Signal

%Linear model
yout(k)=-den(2)*y_1-den(3)*y_2-den(4)*y_3+num(2)*u_1+num(3)*u_2+num(4)*u_3;

D(k)=0.50*rands(1);  %Disturbance signal
yyout(k)=yout(k)+D(k);
```

```
M=2;
if M==1             %No filter
   filty(k)=yyout(k);
elseif M==2         %Using filter with tracker and differentiation
r=2000;
h=0.02;
T=ts;

delta=r*h;
delta0=delta*h;
y=x(1)-yyout(k)+h*x(2);
a0=sqrt(delta*delta+8*r*abs(y));
if abs(y)<=delta0
    a=x(2)+y/h;
else
    a=x(2)+0.5*(a0-delta)*sign(y);
end
if abs(a)<=delta
   fst2=-r*a/delta;
else
   fst2=-r*sign(a);
end

x(1)=x(1)+T*x(2);
x(2)=x(2)+T*fst2;
filty(k)=x(1);
end

error(k)=rin(k)-filty(k);
%I separation
if abs(error(k))<=0.8
   ei=ei+error(k)*ts;
else
   ei=0;
end
    u(k)=kp*error(k)+ki*ei;

if u(k)>=10       % Restricting the output of controller
    u(k)=10;
end
```

```
if u(k <  10
   u(k)  10;
end
%     -    Return of PID parameters-     --
rin_1 rin k);
u 5 u_4;u_4 u_3;u_3-u_2;u_2 u_1,u_1 u(k ;
y 3 y_2;y_2 y_1;y_1 yout(k ;
end
figure(1 ;
subplot(211 ;
plot time,rin, b',time,filty, r' ;
xlabel('time(s    ,ylabel  rin,yout');
subplot(212);
plot(time,u,'r );
xlabel('time(s)',;ylabel('u',;

figure 2 ;
plot time,D, r');
xlabel 'time(s)' ;ylabel('Disturbance signal');

figure 3 ;
plot time,yyout, r',time,filty, b' ;
xlabel  time(s ');ylabel  ideal signal,practical signal');
```

第 2 章　常用的 PID 控制系统

2.1　单回路 PID 控制系统

单回路 PID 控制系统是指系统只有一个 PID 控制器，如图 2-1 所示。本书所述的大部分内容都是关于单回路 PID 控制系统的。

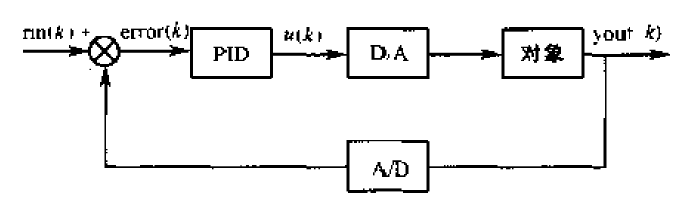


图 2-1　单回路 PID 控制系统

单回路 PID 控制系统的 MATLAB 仿真见第 1 章。

2.2　串级 PID 控制

2.2.1　串级 PID 控制原理

串级计算机控制系统的典型结构如图 2-2 所示，系统中有两个 PID 控制器，$G_{c2}(s)$ 称为副调节器传递函数，包围 $G_{c2}(s)$ 的内环称为副回路。$G_{c1}(s)$ 称为主调节器传递函数，包围 $G_{c1}(s)$ 的外环称为主回路。主调节器的输出控制量 u_1 作为副回路的给定量 $R_2(s)$。

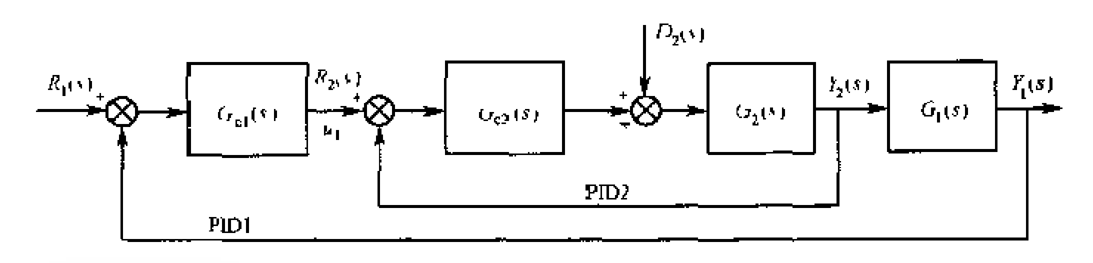


图 2-2　串级控制系统框图

串级控制系统的计算顺序是先主回路（PID1），后副回路（PID2）。控制方式有两种：一种是异步采样控制，即主回路的采样控制周期 T_1 是副回路采样控制周期 T_2 的整数倍。这是因为一般串级控制系统中主控对象的响应速度慢、副控对象的响应速度快的缘故。另一种是同步采样控制，即主、副回路的采样控制周期相同。这时，应根据副回路选择采样周期，因为副回路的受控对象的响应速度较快。

串级控制的主要优点：

（1）将干扰加到副回路中，由副回路控制对其进行抑制；

（2）副回路中参数的变化，由副回路给予控制，对被控量 G_1 的影响大为减弱；

（3）副回路的惯性由副回路给予调节，因而提高了整个系统的响应速度。

副回路是串级系统设计的关键。副回路设计的方式有很多种，下面介绍按预期闭环特性设计副调节器的设计方法。

由副回路框图可得副回路闭环系统的传递函数为：

$$\varphi_2(z)=\frac{Y_2(z)}{U_1(z)}=\frac{G_{c2}(z)G_2(z)}{1+G_{c2}(z)G_2(z)} \tag{2.1}$$

可得副调节器控制律：

$$G_{c2}(z)=\frac{\varphi_2(z)}{G_2(z)(1-\varphi_2(z))} \tag{2.2}$$

一般选择

$$\varphi_2(z)=z^{-n} \tag{2.3}$$

式中，n 为 $G_2(z)$ 有理多项式分母最高次幂。

2.2.2 仿真程序及分析

仿真实例

设副对象特性为 $G_2(s)=1/(T_{02}s+1)$，主对象特性为 $G_1(s)=1/(T_{01}s+1)$，$T_{01}=T_{02}=10$，采样时间为 2s，外加干扰信号为幅度 0.01 的随机信号：

$$d_2(k)=0.01\text{rands}(1)$$

仿真方法一

在离散方式下进行仿真，采用 M 语言进行编程。按预期闭环方法设计副调节器。由于副对象的传递函数为一阶，故由式（2.3）得到副回路闭环系统传递函数：

$$\varphi_2(z)=z^{-1}$$

主调节器采用 PI 控制，取 $k_p=1.2$，$k_i=0.02$，副调节器按控制律式（2.2）设计。副回路输入、输出及跃阶响应结果如图 2-3～图 2-5 所示。

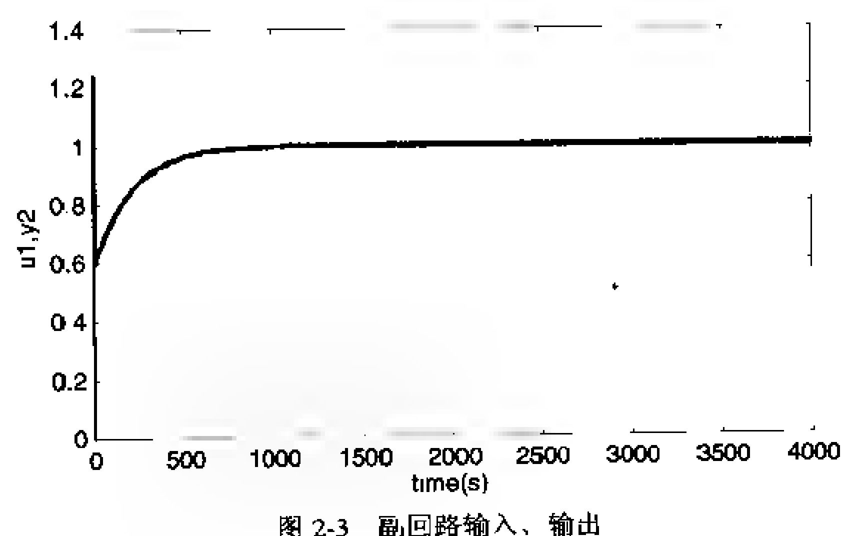


图 2-3　副回路输入、输出

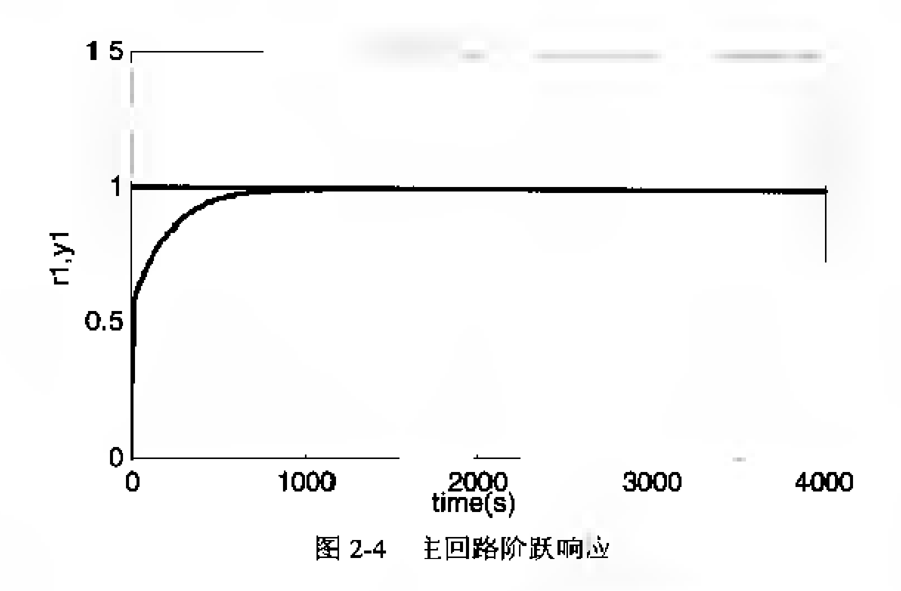


图 2-4　主回路阶跃响应

图 2-5　外加干扰信号

仿真程序: chap2_1.m。

```
%Series System Control
clear all;
close all;

ts=2;
sys1=tf(1,[10,1]);
dsys1=c2d(sys1,ts,'z');
[num1,den1]=tfdata(dsys1,'v');

sys2=tf(1,[10,1]);
dsys2=c2d(sys2,ts,'z');
[num2,den2]=tfdata(dsys2,'v');

dph=1/zpk('z',ts);
Gc2=dph/(dsys2*(1-dph));
[nump,denp]=tfdata(Gc2,'v');
```

```
u1_1=0.0;u2_1=0.0;
y1_1=0;y2_1=0;
e2_1=0;ei=0;

for k=1:1:2000
time(k)=k*ts;

r1(k)=1;
%Linear model
y1(k)=-den1(2)*y1_1+num1(2)*y2_1;  %Main plant

y2(k)=-den2(2)*y2_1+num2(2)*u2_1;  %Assistant plant

error(k)=r1(k)-y1(k);
ei=ei+error(k);
u1(k)=1.2*error(k)+0.02*ei;   %Main Controller

e2(k)=u1(k)-y2(k);            %Assistant Controller
u2(k)=-denp(2)*u2_1+nump(1)*e2(k)+nump(2)*e2_1;

d2(k)=0.01*rands(1);
u2(k)=u2(k)+d2(k);

%----      Return of PID parameters      ---
u1_1=u1(k);
u2_1=u2(k);

e2_1=e2(k);

y1_1=y1(k);
y2_1=y2(k);
end
figure(1);     %Assistant Control
plot(time,u1,'b',time,y2,'r');
xlabel('time(s)');ylabel('u1,y2');

figure(2);     %Main Control
plot(time,r1,'b',time,y1,'r');
xlabel('time(s)');ylabel('r1,y1');

figure(3);
```

仿真方法二

按串级控制的基本原理，采用 Simulink 进行编程，在连续方式下进行仿真，仿真程序如图 2-6 所示。在串级控制中，主调节器采用 PI 控制，取 $k_p = 50, k_i = 5$，副调节器采用 P 控制，$k_p = 200$。外加干扰为正弦信号 $\sin(50t)$，通过切换开关的切换，分别实现常规 PID 控制和串级控制，它们的阶跃响应结果如图 2-7 和图 2-8 所示。



图 2-6　串级控制仿真程序

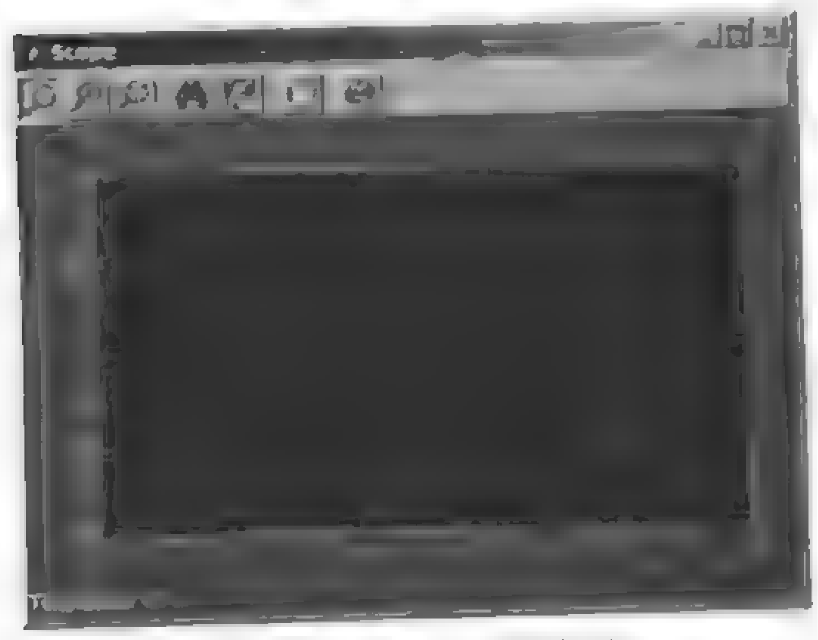


图 2-7　常规 PID 控制的阶跃响应

仿真程序：chap2_2.mdl，如图 2-8 所示。

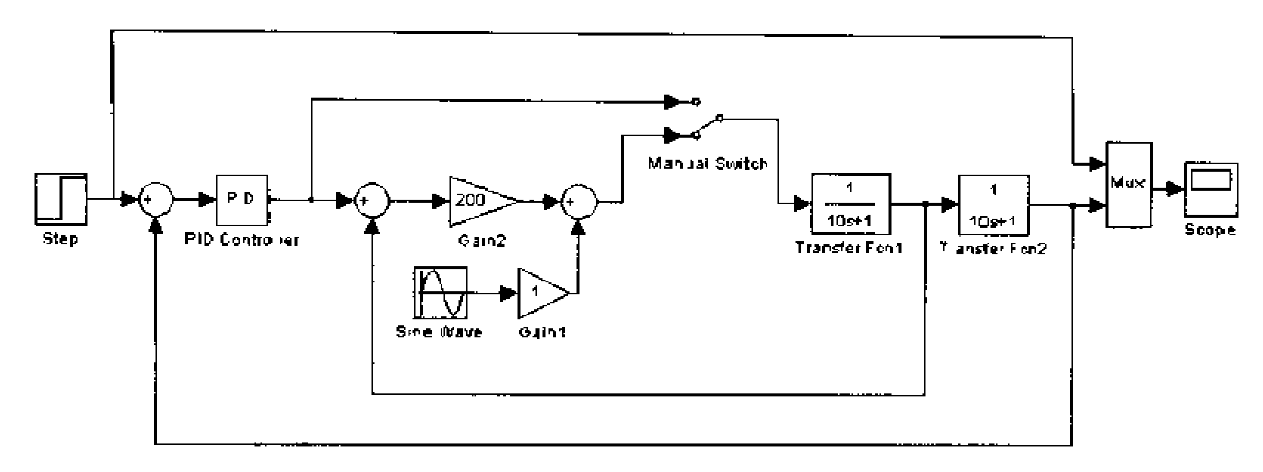


图 2-8　串级控制的 Simulink 仿真程序

2.3　纯滞后系统的大林控制算法

2.3.1　大林控制算法原理

早在 1968 年，美国 IBM 公司的大林（Dahlin）就提出一种不同于常规 PID 控制规律的新型算法，即大林控制算法。该算法的最大特点是，将期望的闭环响应设计成一阶惯性加纯延迟，然后反过来得到能满足这种闭环响应的控制器。

对于如图 2-9 所示的单回路控制系统，$G_c(z)$ 为数字控制器，$G_p(z)$ 为被控对象，则闭环系统传递函数为：

$$\phi(z)=\frac{Y(z)}{R(z)}=\frac{G_c(z)G_p(z)}{1+G_c(z)G_p(z)} \tag{2.4}$$

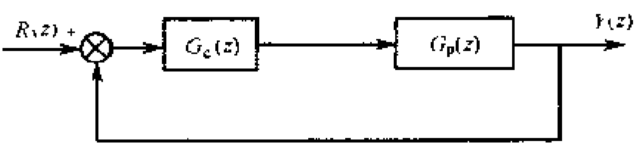


图 2-9　单回路控制系统框图

则有：

$$G_c(z)=\frac{U(z)}{E(z)}=\frac{1}{G_p(z)}\frac{\phi(z)}{1-\phi(z)} \tag{2.5}$$

如果能事先设定系统的闭环响应 $\phi(z)$，则可得控制器 $G_c(z)$。大林指出，通常的期望闭环响应是一阶惯性加纯延迟形式，其延迟时间等于对象的纯延迟时间 τ：

$$\phi(s)=\frac{Y(s)}{R(s)}=\frac{e^{-\tau s}}{T_\varphi s+1} \tag{2.6}$$

式中，T_φ 为闭环系统的时间常数，由此而得到的控制律称为大林控制算法。

2.3.2　仿真程序及分析

仿真实例

设被控对象为：

$$G_p(s)=\frac{e^{-0.76s}}{0.4s+1}$$

采样时间为0.5s，期望的闭环响应设计为：

$$\phi(s)=\frac{Y(s)}{R(s)}=\frac{e^{-0.76s}}{0.15s+1}$$

$M=1$时为采用大林控制算法；$M=2$时为采用普通PID控制算法。可见，采用大林控制算法可取得很好的控制效果，其阶跃响应结果如图2-10和图2-11所示。

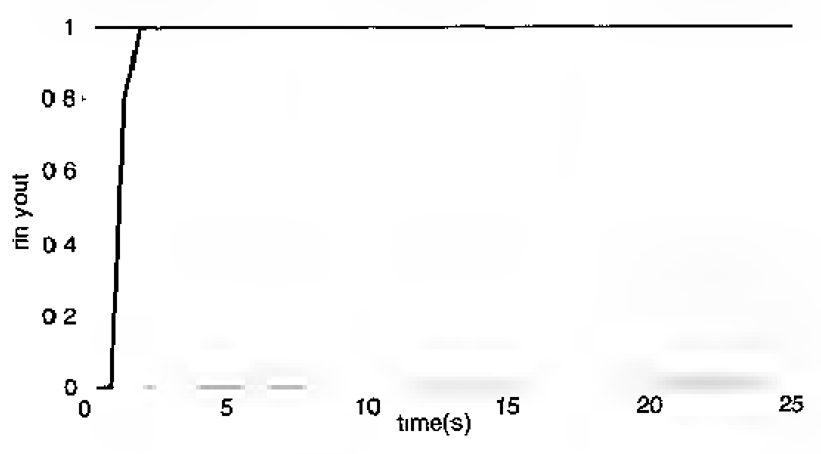


图2-10　大林控制算法阶跃响应（M=1）

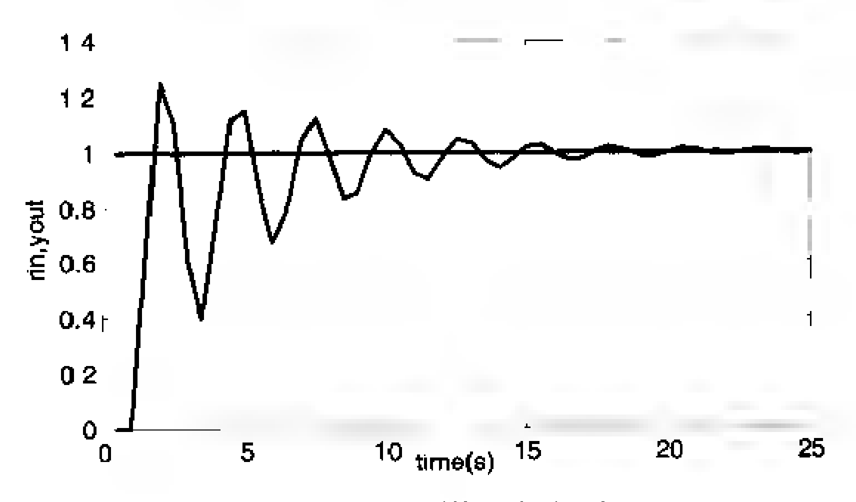


图2-11　普通PID控制算法阶跃响应（M=2）

仿真程序: chap2_3.m。

```
%Delay Control with Dalin Algorithm
clear all;
close all;
ts=0.5;

%Plant
sys1=tf([1],[0.4,1],'inputdelay',0.76);
dsys1=c2d(sys1,ts,'zoh');
[num1,den1]=tfdata(dsys1,'v');
```

```
%Ideal closed loop
sys2=tf([1],[0.15,1],'inputdelay',0.76);
dsys2=c2d(sys2,ts,'zoh');

%Design Dalin controller
dsys=1/dsys1*dsys2/(1-dsys2);
[num,den]=tfdata(dsys,'v');

u_1=0.0;u_2=0.0;u_3=0.0;u_4=0.0;u_5=0.0;
y_1=0.0;

error_1=0.0;error_2=0.0;error_3=0.0;
ei=0;
for k=1:1:50
time(k)=k*ts;

rin(k)=1.0;  %Tracing Step Signal

yout(k)=-den1(2)*y_1+num1(2)*u_2+num1(3)*u_3;
error(k)=rin(k)-yout(k);

M=1;
if M==1        %Using Dalin Method
u(k)=(num(1)*error(k)+num(2)*error_1+num(3)*error_2+num(4)*error_3...
     -den(3)*u_1-den(4)*u_2-den(5)*u_3-den(6)*u_4-den(7)*u_5)/den(2);
elseif M==2    %Using PID Method
ei=ei+error(k)*ts;
u(k)=1.0*error(k)+0.10*(error(k)-error_1)/ts+0.50*ei;
end
%----------Return of dalin parameters----------
u_5=u_4;u_4=u_3;u_3=u_2;u_2=u_1;u_1=u(k);
y_1=yout(k);

error_3=error_2;error_2=error_1;error_1=error(k);
end
plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('rin,yout');
```

2.4 纯滞后系统的 Smith 控制算法

在工业过程控制中，许多被控对象具有纯滞后的性质。Smith（史密斯）提出了一种纯滞后补偿模型，其原理为，与 PID 控制器并接一个补偿环节，该补偿环节称为 Smith 预估器。

2.4.1 连续 Smith 预估控制

带有纯延迟的单回路控制系统如图 2-12 所示，其闭环传递函数为：

$$\phi(s)=\frac{Y(s)}{R(s)}=\frac{G_c(s)G_0(s)e^{-\tau s}}{1+G_c(s)G_0(s)e^{-\tau s}} \tag{2.7}$$

其特征方程为：

$$1+G_c(s)G_0(s)e^{-\tau s}=0 \tag{2.8}$$

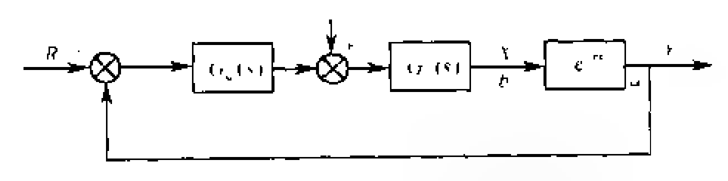


图 2-12 带有纯延迟的单回路控制系统

可见，特征方程中出现了纯延迟环节，使系统稳定性降低，如果 τ 足够大，系统将不稳定，这就是大延迟过程难于控制的本质。而 $e^{-\tau s}$ 之所以在特征方程中出现，是由于反馈信号是从系统的 a 点引出来的，若能将反馈信号从 b 点引出，则把纯延迟环节移到控制回路的外边，如图 2-13 所示，经过 τ 的延迟时间后，被调量 Y 将重复 X 同样的变化。

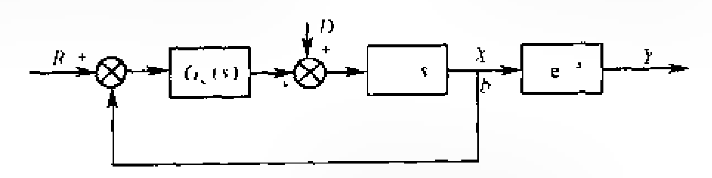


图 2-13 改进的有纯延迟的单回路控制系统

由于反馈信号 X 没有延迟，系统的响应会大大改善，但在实际系统中，b 点或是不存在，或是受物理条件的限制，无法从 b 点引出反馈信号来。针对这种问题，Smith 提出采用人造模型的方法，构造如图 2-14 所示的控制系统。

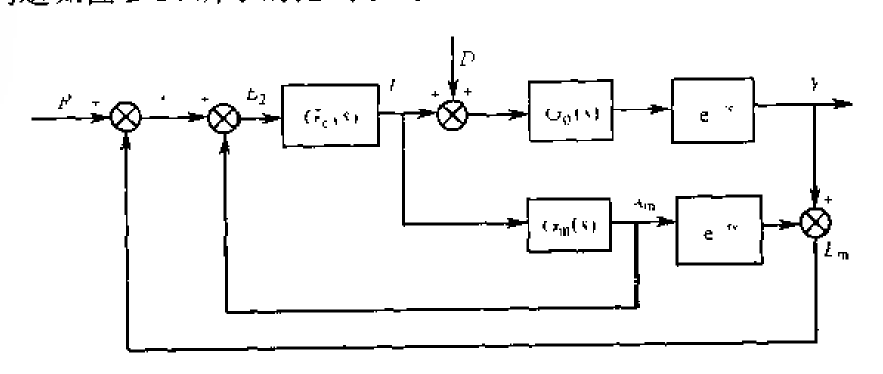


图 2-14 Smith 预估控制系统

如果模型是精确的，即 $G_0(s)=G_m(s), \tau=\tau_m$，且不存在负荷扰动 $(D=0)$，则 $Y=Y_m$，$E_m=Y-Y_m=0$，$X=X_m$，则可以用 X_m 代替 X 作第一条反馈回路，实现将纯延迟环节移到控制回路的外边。如果模型是不精确的或是出现负荷扰动，则 X 就不等于 X_m，$E_m=Y-Y_m\neq 0$，控制精度也就不能令人满意。为此，采用 E_m 实现第二条反馈回路。这就是 Smith 预估器的控制策略。

实际上预估模型不是并联在过程上，而是反向并联在控制器上的，因此，将图 2-14 变换得到 Smith 预估控制系统等效图，如图 2-15 所示。

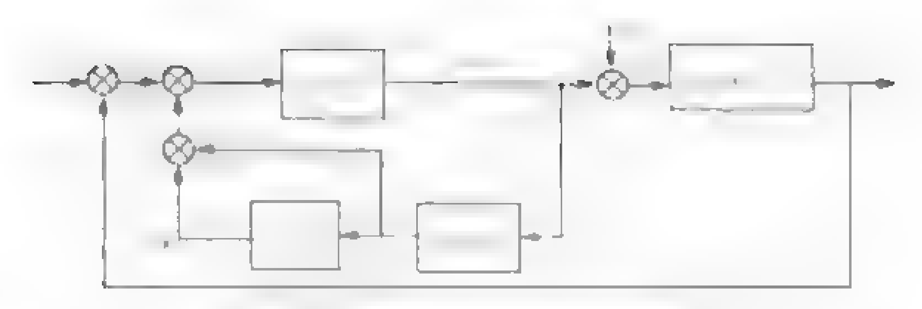


图 2-15 Smith 预估控制系统等效图

显然，Smith 控制方法的前提是必须确切地知道被控对象的数学模型，在此基础上才能建立精确的预估模型。

2.4.2 仿真程序及分析

仿真实例

设被控制对象为：

$$G_p(s)=\frac{e^{-80s}}{60s+1}$$

采用 Smith 控制方法，在 PI 控制中，取 $k_p=4.0$，$k_i=0.022$，假设预测模型精确，阶跃指令信号取 100。Simulink 仿真程序及仿真结果如图 2-16～图 2-18 所示。仿真结果表明，Smith 控制方法具有很好的控制效果。

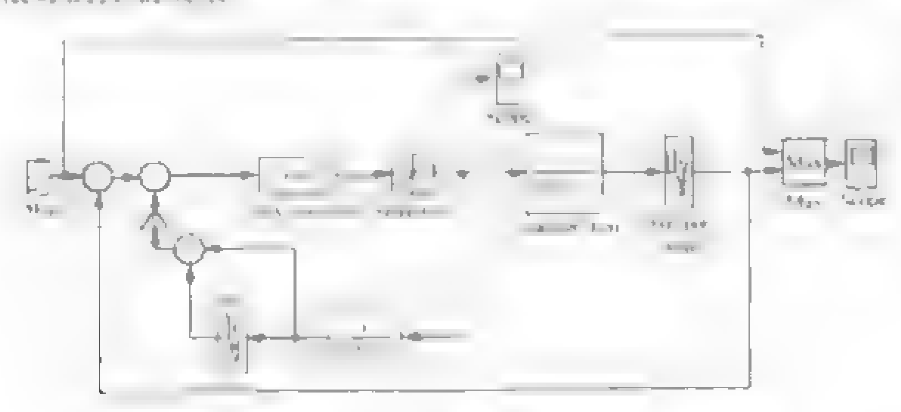


图 2-16 Simulink 仿真程序

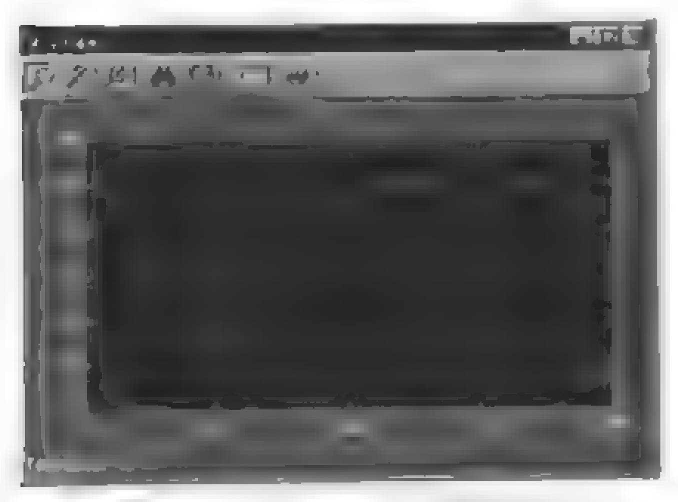


图 2-17 不用 Smith 补偿的阶跃响应

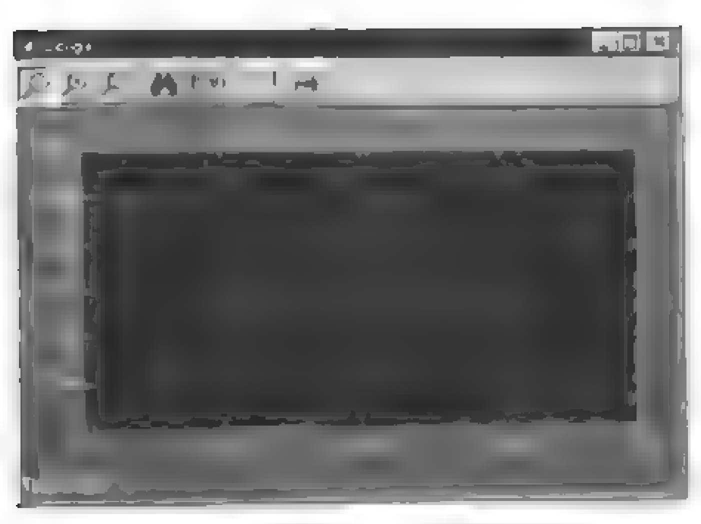


图 2-18 采用 Smith 补偿的阶跃响应

仿真程序：chap2_4.mdl，如图 2-13 所示。

2.4.3 数字 Smith 预估控制

由于控制器一般采用计算机实现，因此有必要研究数字 Smith 预估控制的仿真。设被控对象的传递函数为：

$$G_p(s)=\frac{k_p e^{-\tau s}}{T_1 s+1}=G_c(s)e^{-\tau s} \tag{2.9}$$

数字 Smith 预估控制系统框图如图 2-19 所示。

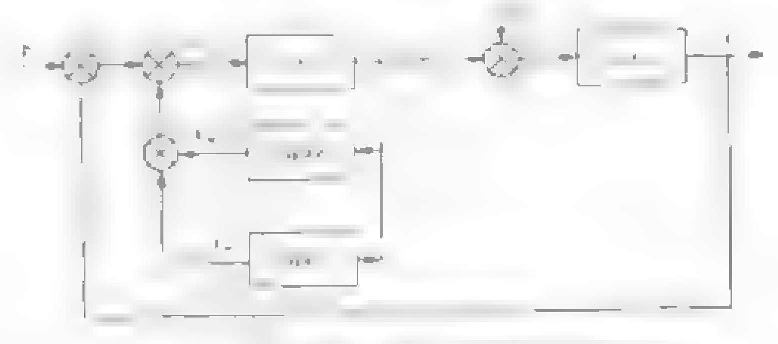


图 2-19 数字 Smith 预估控制系统框图

由图 2-19 可得：

$$e_2(k)=e_1(k)-x_m(k)+y_m(k)=r(k)-y(k)-x_m(k)+y_m(k) \tag{2.10}$$

若模型是精确的，则有：

$$y(k)=y_m(k) \tag{2.11}$$

$$e_2(k)=r(k)-x_m(k) \tag{2.12}$$

$e_2(k)$ 为数字控制器 $G_c(z)$ 的输入，$G_c(z)$ 一般采用 PI 控制算法。

2.4.4 仿真程序及分析

仿真实例

设被控对象为：

$$G_p(s)=\frac{e^{-80s}}{60s+1}$$

采样时间为20s 。

仿真方法一

采用M语言进行数字化仿真。按Smith算法设计控制器。S代表指令信号的类型，S=1为阶跃响应，S=2为方波响应，M代表三种情况下的仿真：$M=1$为模型不精确，$M=2$为模型精确，$M=3$为采用PI控制。取$S=2$，针对$M=1$，$M=2$，$M=3$三种情况进行仿真。在PI控制中，$k_p=0.50, k_i=0.010$。其响应结果如图2-20～图2-22所示。

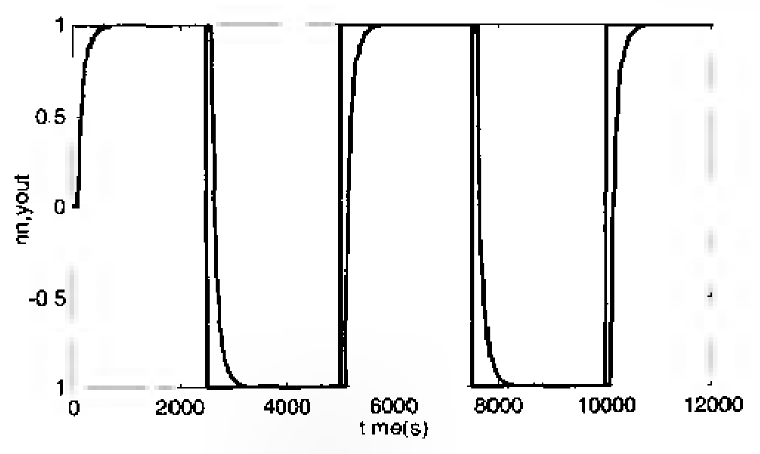


图2-20　模型不精确时方波响应（M=1）

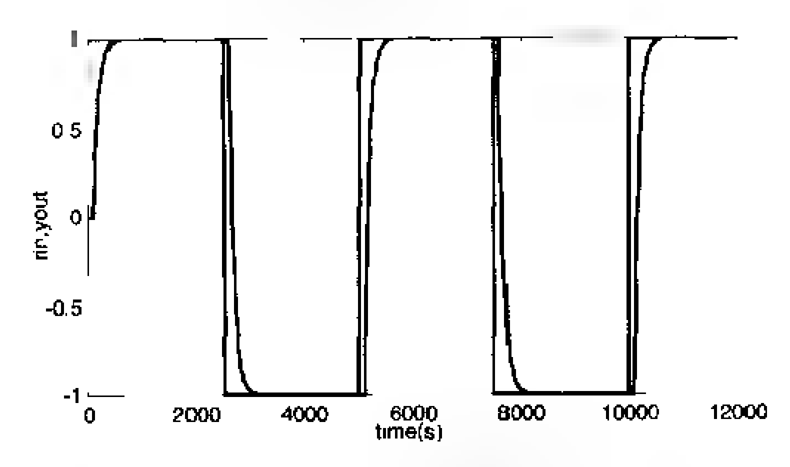


图2-21　模型精确时方波响应（M=2）

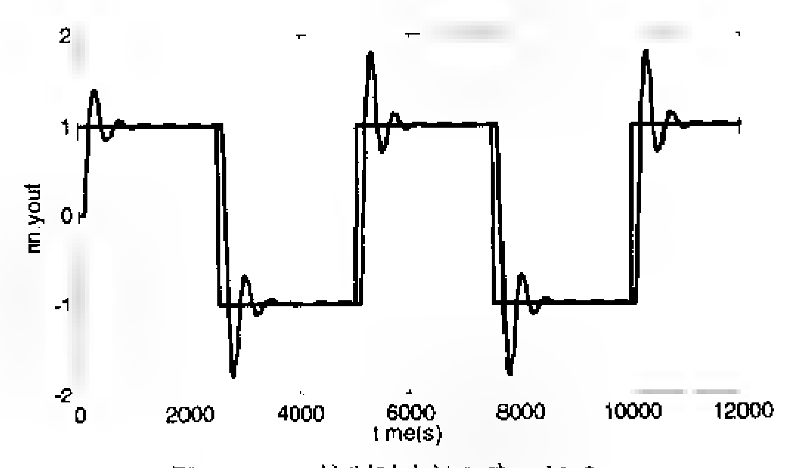


图2-22　PI控制时方波响应（M=3）

仿真程序清单：chap2_5.m。

```
%Big Delay PID Control with Smith Algorithm
clear all;close all;
Ts=20;

%Delay plant
kp=1;
Tp=60;
tol=80;
sys=tf([kp],[Tp,1],'inputdelay',tol);
dsys=c2d(sys,Ts,'zoh');
[num,den]=tfdata(dsys,'v');

M=1;
%Prediction model
if M==1  %No Precise Model: PI+Smith
   kp1=kp*1.10;
   Tp1=Tp*1.10;
   tol1=tol*1.0;
elseif M==2|M==3  %Precise Model: PI+Smith
   kp1=kp;
   Tp1=Tp;
   tol1=tol;
end

sys1=tf([kp1],[Tp1,1],'inputdelay',tol1);
dsys1=c2d(sys1,Ts,'zoh');
[num1,den1]=tfdata(dsys1,'v');

u_1=0.0;u_2=0.0;u_3=0.0;u_4=0.0;u_5=0.0;
e1_1=0;
e2=0.0;
e2_1=0.0;
ei=0;

xm_1=0.0;
ym_1=0.0;
y_1=0.0;

for k=1:1:600
```

```
   time(k)=k*Ts;

S=2;
if S==1
   rin(k)=1.0;     %Tracing Step Signal
end
if S==2
   rin(k)=sign(sin(0.0002*2*pi*k*Ts)); %Tracing Square Wave Signal
end

%Prediction model
xm(k)=-den1(2)*xm_1+num1(2)*u_1;
ym(k)=-den1(2)*ym_1+num1(2)*u_5;  %With Delay

yout(k)=-den(2)*y_1+num(2)*u_5;

if M==1        %No Precise Model: PI+Smith
   e1(k)=rin(k)-yout(k);
   e2(k)=e1(k)-xm(k)+ym(k);
     ei=ei+Ts*e2(k);
     u(k)=0.50*e2(k)+0.010*ei;
   e1_1=e1(k);
elseif M==2   %Precise Model: yout(k)=ym(k),  PI+Smith
   e2(k)=rin(k)-xm(k);
     ei=ei+Ts*e2(k);
     u(k)=0.50*e2(k)+0.010*ei;
     e2_1=e2(k);
elseif M==3  %Only PI
   e1(k)=rin(k)-yout(k);
     ei=ei+Ts*e1(k);
     u(k)=0.50*e1(k)+0.010*ei;
   e1_1=e1(k);
end

%----------Return of smith parameters----------
xm_1=xm(k);
ym_1=ym(k);

u_5=u_4;u_4=u_3;u_3=u_2;u_2=u_1;u_1=u(k);
y_1=yout(k);
```

方法二

采用 Simulink 进行数字化仿真，按 Smith 算法设计 Simulink 模块，在 PI 控制中 $k_p=4.0, k_i=0.022$，其响应结果如图 2-23 和图 2-24 所示。



图 2-23　Smith 阶跃响应结果

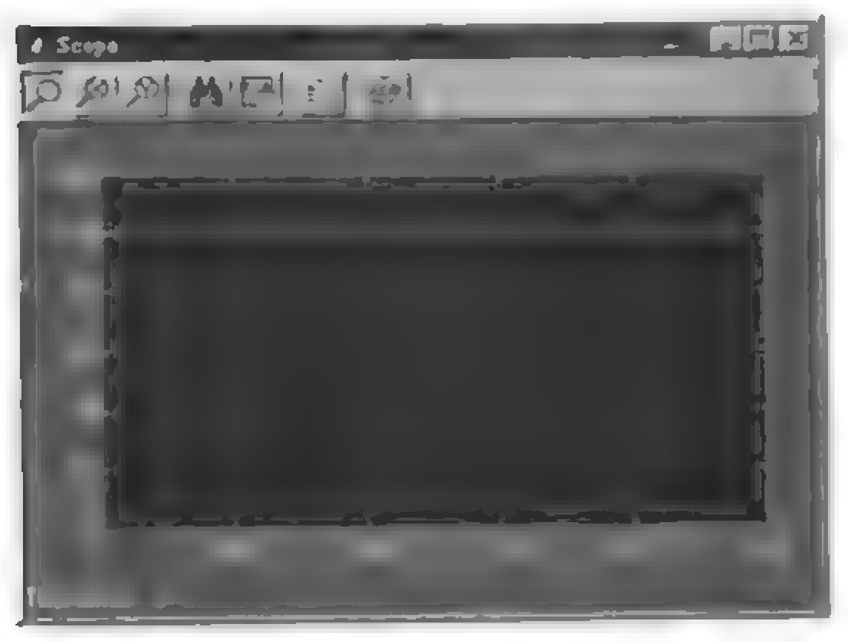


图 2-24　不加入 Smith 时阶跃响应结果

仿真程序：chap2_6.mdl，如图 2-25 所示。

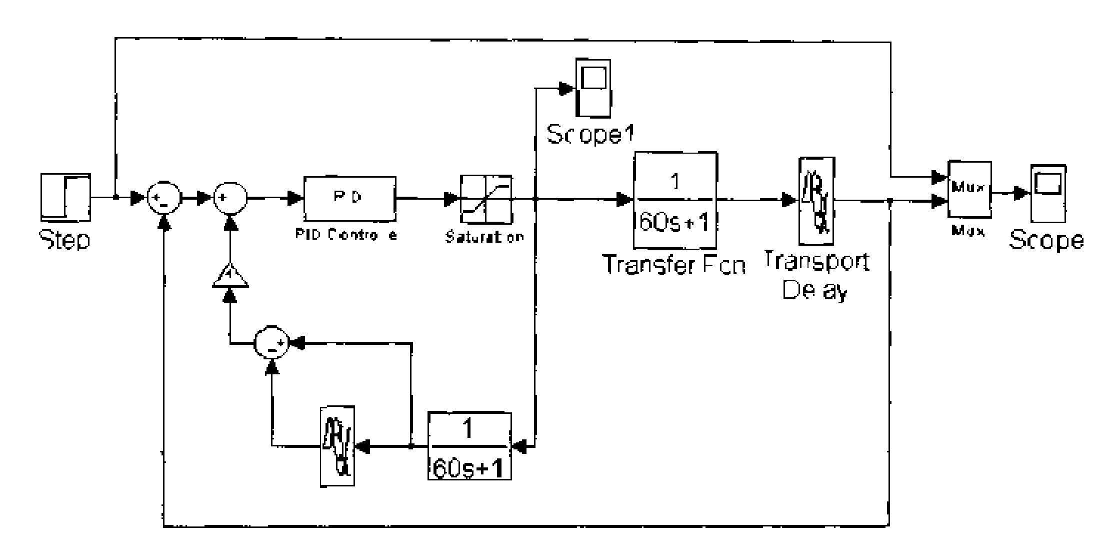


图 2-25 数字 Smith 预估控制的 Simulink 仿真

2.5 基于 Ziegler-Nichols 方法的 PID 整定

2.5.1 连续 Ziegler-Nichols 方法的 PID 整定

Ziegler-Nichols 方法是基于稳定性分析的 PID 整定方法。该方法整定比例系数 K_P 的思想是，首先置 $K_D = K_I = 0$，然后增加 K_P 直至系统开始振荡（即闭环系统极点在 $j\omega$ 轴上），再将 K_P 乘以 0.6，即为整定后的比例系数 K_P。

整定公式如下：

$$K_P = 0.6K_m,\quad K_D = \frac{K_P\pi}{4\omega_m},\quad K_I = \frac{K_P\omega_m}{\pi} \tag{2.13}$$

式中，K_m 为系统开始振荡时的 K 值，ω_m 为振荡频率。

利用根轨迹法可以确定 K_m 和 ω_m。对于给定的被控对象传递函数，可以得到其根轨迹。对应穿越 $j\omega$ 轴时的增益即为 K_m，而此点的 ω 值即为 ω_m。

2.5.2 仿真程序及分析

仿真实例

设被控对象为：

$$G(s) = \frac{400}{s(s^2 + 30s + 200)}$$

使用 rlocus 及 rlocfind 命令可求得穿越增益 K_m=14 和穿越频率 ω_m =14rad/s。采用 Ziegler-Nichols 整定方法式（2.13）可求得 PID 参数：

$$K_P = 8.8371,\quad K_D = 0.4945,\quad K_I = 39.4847$$

运行整定程序 chap2_7f.m，可得图 2-26～图 2-28。图 2-26 示出系统未补偿的根轨迹图，在该图上可选定穿越 $j\omega$ 轴时的增益 K_m 和该点的 ω 值，即 ω_m。整定程序中，sys_pid 和 sysc 分别为控制器和闭环系统的传递函数。图 2-27 示出整定前后系统的伯特图，可见该系统整定后，频带拓宽，相移超前。图 2-28 示出整定后系统的根轨迹，所有极点位于负半面，达到完

全稳定状态。

运行 Simulink 控制程序 chap2_7.mdl，通过开关切换进行两种方法的仿真，可得图 2-29 和图 2-30，图 2-29 示出系统未补偿的正弦跟踪，图 2-30 示出系统采用 PID 补偿后的正弦跟踪。

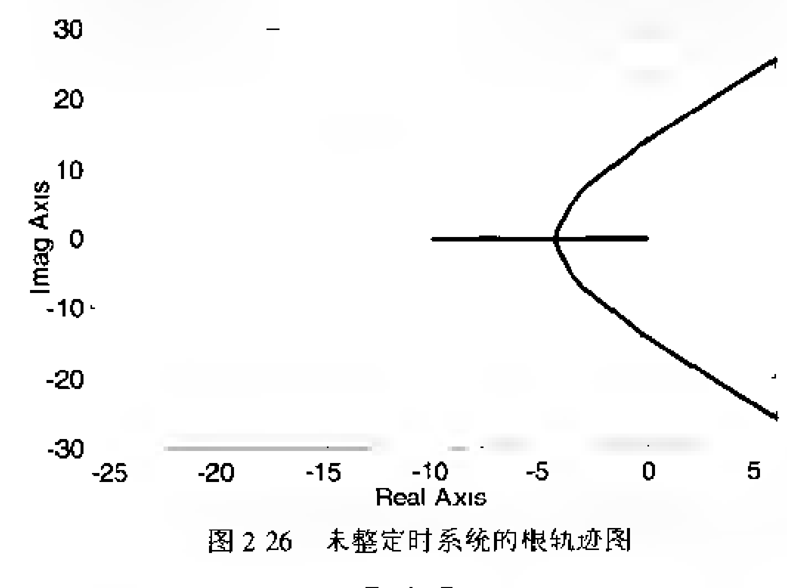


图 2-26　未整定时系统的根轨迹图

图 2-27　整定前后系统的伯特图（实线为整定前，虚线为整定后）

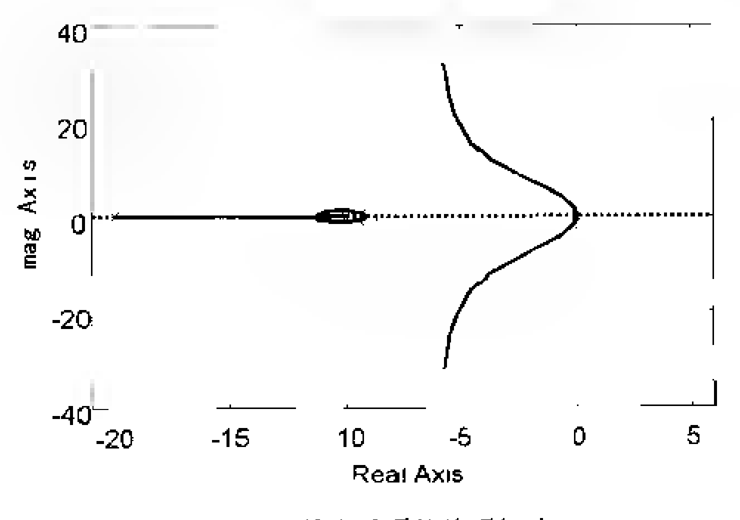


图 2-28　整定后系统的根轨迹

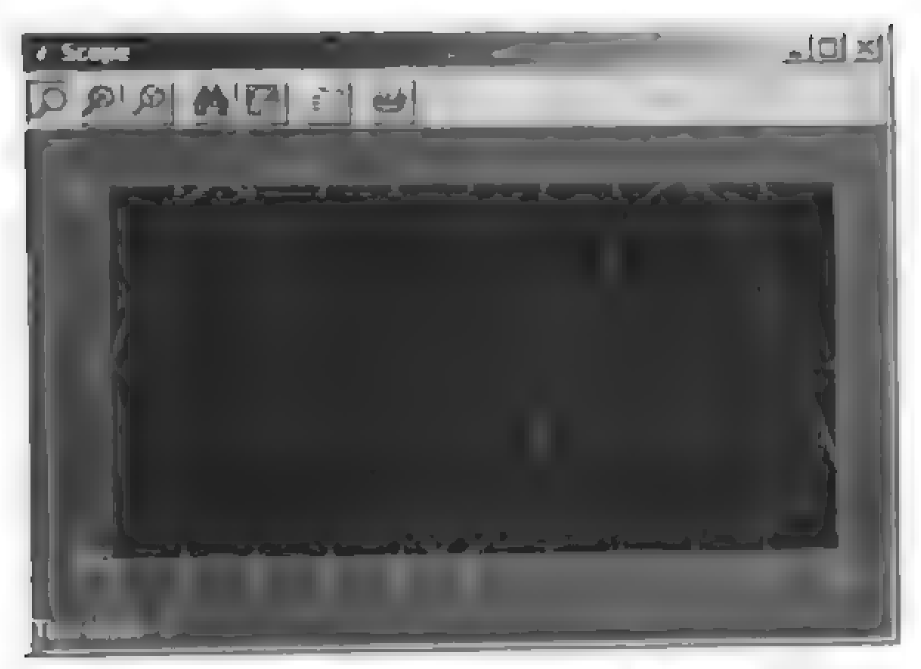


图 2-29　整定前的正弦跟踪

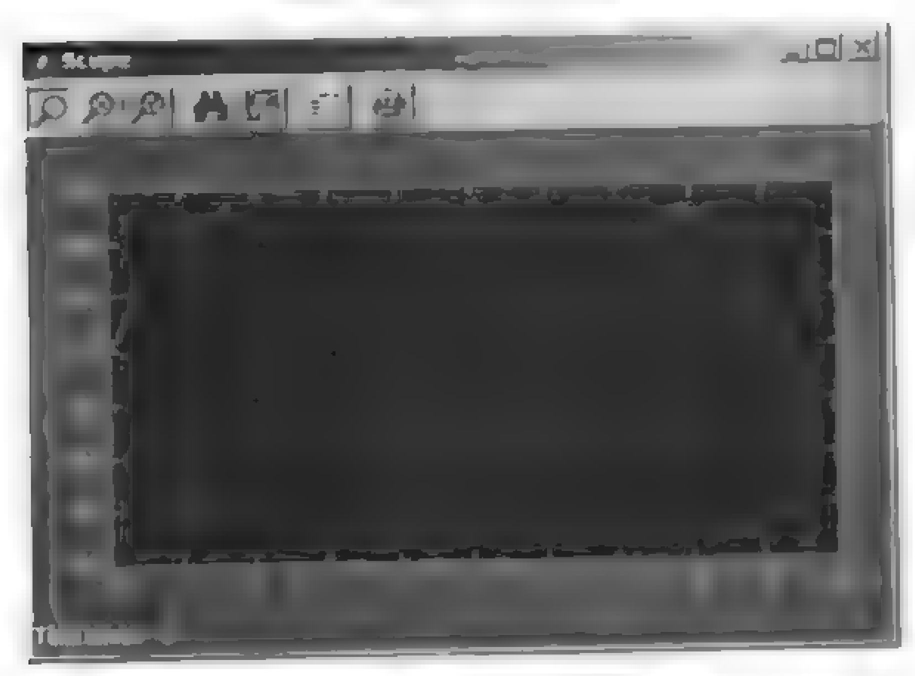


图 2-30　整定后的正弦跟踪

仿真程序分为 PID 整定程序和 Simulink 仿真程序两部分

整定程序：chap2_7f.m

```
wm=imag(pole(2));
kp=0.6*km
kd=kp*pi/(4*wm)
ki=kp*wm/pi

figure(2);
grid on;
bode(sys,'r');

sys_pid=tf([kd,kp,ki],[1,0])
sysc=series(sys,sys_pid)
hold on;
bode(sysc,'b')

figure(3);
rlocus(sysc);
```

Simulink 控制程序：chap2_7.mdl，如图 2-31 所示。

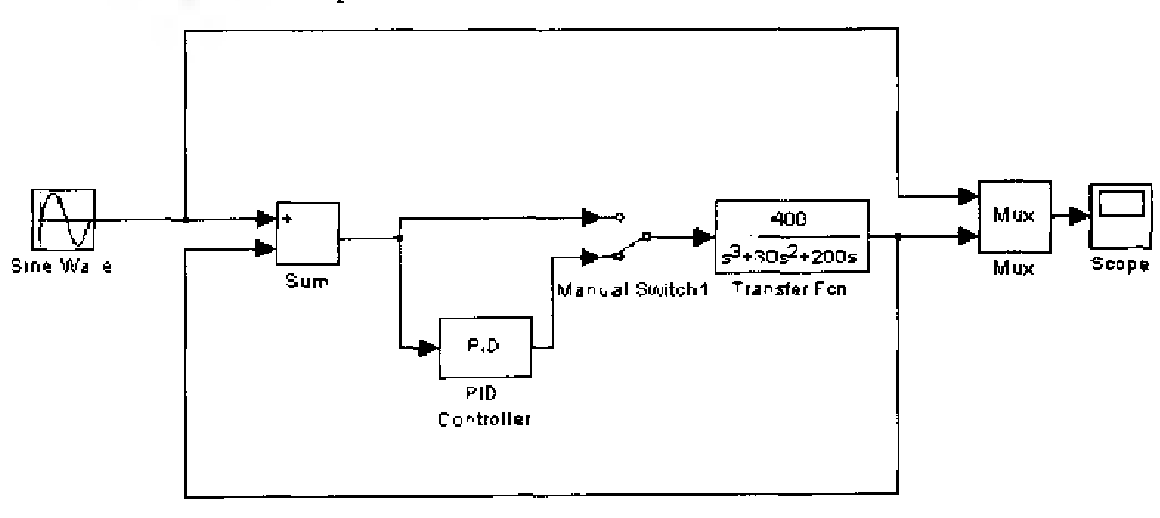


图 2-31　Simulink 仿真程序

2.5.3　离散 Ziegler-Nichols 方法的 PID 整定

Ziegler-Nichols 方法同样适用于离散系统的 PID 整定。该方法整定比例系数 K_P 的思想是，首先置 K_D=K_I=0，然后增加 K_P 直至系统开始振荡（即使系统的闭环极点位于 z 平面的单位圆上），将所得的 K_P 乘以 0.6，即为整定后的比例系数 K_P。

整定公式如下：

$$K_P = 0.6K_m,\quad K_D = \frac{K_P \pi}{4\omega_m},\quad K_I = \frac{K_P \omega_m}{\pi} \tag{2.14}$$

式中，K_m为系统开始振荡时的 K 值，ω_m 为振荡频率。振荡频率 ω_m 可以由极点位于单位圆上的角度θ得到，$\omega_m=\theta/T$（T 为采样周期）。

2.5.4 仿真程序及分析

仿真实例

设被控制对象为：

$$G(s)=\frac{1}{10s^2+2s}$$

采样周期为 T=0.25s。

采用零阶保持器将对象离散化，使用 rlocus 及 rlocfind 命令给出 $G(z)$的根轨迹图，可求得振荡增益 K_m=11.2604 和振荡频率 ω_m=1.0546 rad/s。采用 Ziegler-Nichols 方法，由式（2.14）可求得离散 PID 参数：

$$K_P=6.7562\text{，}\quad K_D=5.0318\text{，}\quad K_I=2.2679$$

运行整定程序 chap2_8f.m，可得图 2-32 和图 2-33。图 2-32 示出系统未补偿的根轨迹图，在该图上选定位于 z 平面单位圆上的闭环极点，则求得所对应的增益 K_m和该点对应的ω_m。

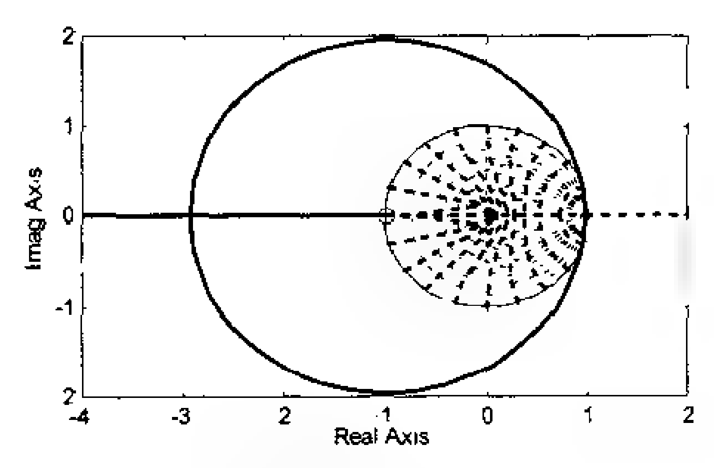


图 2-32　未整定时系统的根轨迹图

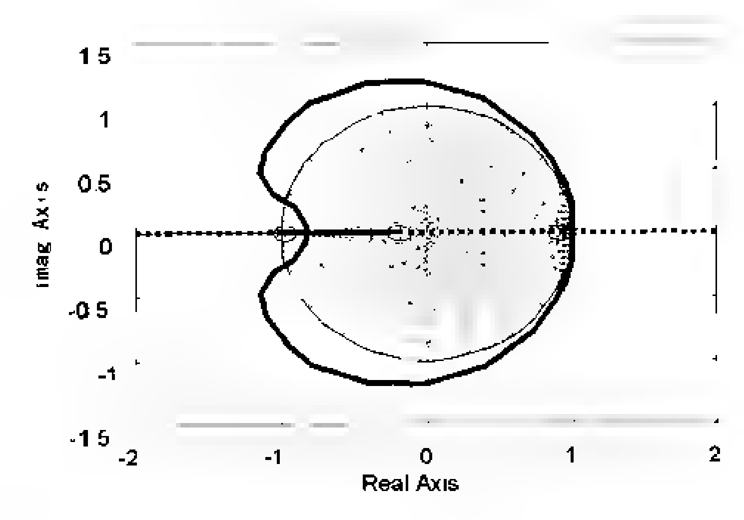


图 2-33　整定后系统的根轨迹图

整定程序中，dsys_pid 和 dsysc 分别为离散的控制器及校正后的离散闭环系统。图 2-33

示出 PID 整定后系统的根轨迹。

运行控制程序 chap2_8.m，通过开关切换 M 进行两种方法的仿真，可得图 2-34 和图 2-35，图 2-34 示出系统未补偿的正弦跟踪，图 2-35 示出系统采用 PID 补偿后的正弦跟踪。

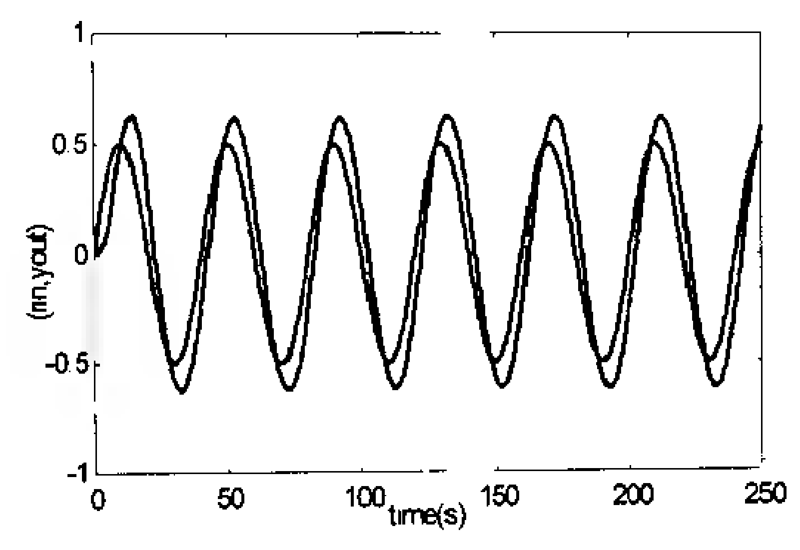


图 2-34 整定前的正弦跟踪 ，M=2

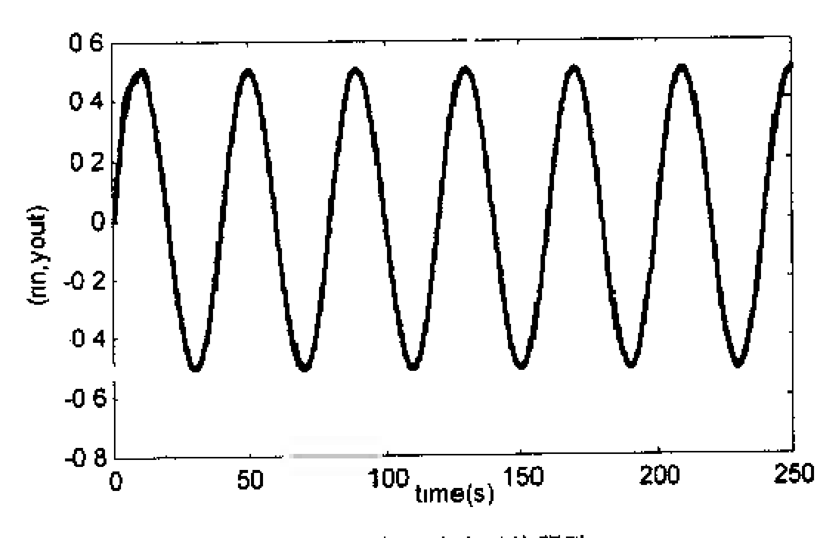


图 2-35 整定后的正弦跟踪

仿真程序分为 PID 整定程序和 PID 控制程序两部分。

PID 整定程序：chap2_8f.m。

```
%PID Controller Based on Ziegler Nichols
clear all;
close all;

ts=0.25;
sys=tf(1,[10,2,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');
```

```
axis('square'),zgrid('new');

figure(1);
rlocus(dsys);
[km,pole]=rlocfind(dsys)

wm=angle(pole(1))/ts;
kp=0.6*km
kd=kp*pi/(4*wm)
ki=kp*wm/pi

dsys_pid=kp+kd*tf([1,-1],[1,1],ts)+ki*tf([1,0],[1,-1],ts)*ts;
dsysc=dsys*dsys_pid;
figure(2);
rlocus(dsysc);
axis('square'),zgrid;
```

PID 控制程序：chap2_8.m。

```
%PID Controller 2001 9 6
close all;

ts=0.25;
sys=tf(1,[10,2,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

u_1=0;u_2=0;
y_1=0;y_2=0;

x=[0,0,0]';

error_1=0;

for k=1:1:1000
time(k)=k*ts;

%rin(k)=1.0;
rin(k)=0.5*sin(0.025*2*pi*k*ts);

%Linear model
```

```
yout(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;
error(k)=rin(k)-yout(k);

x(1)=error(k);                     % Calculating P
x(2)=(error(k)-error_1)/ts;        % Calculating D
x(3)=x(3)+error(k)*ts;             % Calculating I

M=1;
switch M
  case 1       %Using PID
    u(k)=kp*x(1)+kd*x(2)+ki*x(3);
  case 2       %No PID
  u(k)=error(k);
end

u_2=u_1;
u_1=u(k);

y_2=y_1;
y_1=yout(k);

error_1=error(k);
end
figure(1);
plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('(rin,yout)');

figure(2);
plot(time,rin-yout,'r');
xlabel('time(s)');ylabel('error');
```

第 3 章　专家 PID 控制和模糊 PID 控制

3.1　专家 PID 控制

3.1.1　专家 PID 控制原理

专家控制（Expert Control）的实质是基于受控对象和控制规律的各种知识，并以智能的方式利用这些知识来设计控制器。利用专家经验来设计 PID 参数便构成专家 PID 控制。

典型的二阶系统单位阶跃响应误差曲线如图 3-1 所示。对于典型的二阶系统阶跃响应过程进行如下分析。

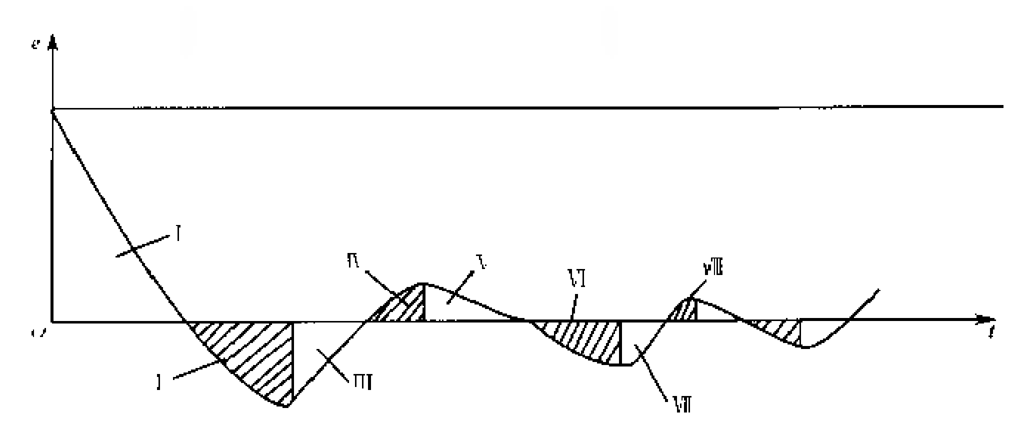


图 3-1　典型二阶系统单位阶跃响应误差曲线

令$e(k)$表示离散化的当前采样时刻的误差值，$e(k-1)$、$e(k-2)$分别表示前一个和前两个采样时刻的误差值，则有：

$$
\begin{aligned}
\Delta e(k) &= e(k)-e(k-1) \\
\Delta e(k-1) &= e(k-1)-e(k-2)
\end{aligned}
\tag{3.1}
$$

根据误差及其变化，可设计专家 PID 控制器，该控制器可分为以下五种情况进行设计：

（1）当$|e(k)|>M_1$时，说明误差的绝对值已经很大。不论误差变化趋势如何，都应考虑控制器的输出应按最大（或最小）输出，以达到迅速调整误差，使误差绝对值以最大速度减小。此时，它相当于实施开环控制。

（2）当$e(k)\Delta e(k)\geq 0$时，说明误差在朝误差绝对值增大方向变化，或误差为某一常值，未发生变化。此时，如果$|e(k)|\geq M_2$，说明误差也较大，可考虑由控制器实施较强的控制作用，以达到扭转误差绝对值朝减小方向变化，并迅速减小误差的绝对值，控制器输出可为：

$$
u(k)=u(k-1)+k_1\left\{k_p[e(k)-e(k-1)]+k_i e(k)+k_d[e(k)-2e(k-1)+e(k-2)]\right\} \tag{3.2}
$$

此时，如果$|e(k)|<M_2$，则说明尽管误差朝绝对值增大方向变化，但误差绝对值本身并

不很大，可考虑控制器实施一般的控制作用，只要扭转误差的变化趋势，使其朝误差绝对值减小方向变化，控制器输出为：

$$u(k)=u(k-1)+k_p[e(k)-e(k-1)]+k_i e(k)+k_d[e(k)-2e(k-1)+e(k-2)] \tag{3.3}$$

（3）当$e(k)\Delta e(k)<0$、$\Delta e(k)\Delta e(k-1)>0$或者$e(k)=0$时，说明误差的绝对值朝减小的方向变化，或者已经达到平衡状态。此时，可考虑采取保持控制器输出不变。

（4）当$e(k)\Delta e(k)<0$、$\Delta e(k)\Delta e(k-1)<0$时，说明误差处于极值状态。如果此时误差的绝对值较大，即$|e(k)|\geqslant M_2$，可考虑实施较强的控制作用：

$$u(k)=u(k-1)+k_1 k_p e_m(k) \tag{3.4}$$

如果此时误差的绝对值较小，即$|e(k)|<M_2$，可考虑实施较弱的控制作用：

$$u(k)=u(k-1)+k_2 k_p e_m(k) \tag{3.5}$$

（5）当$|e(k)|\leqslant\varepsilon$时，说明误差的绝对值很小，此时加入积分，减少稳态误差。

式中，$e_m(k)$——误差e的第k个极值；

$u(k)$——第k次控制器的输出；

$u(k-1)$——第$k-1$次控制器的输出；

k_1——增益放大系数，$k_1>1$；

k_2——抑制系数，$0<k_2<1$；

M_1, M_2——设定的误差界限，$M_1>M_2$；

k——控制周期的序号（自然数）；

ε——任意小的正实数。

在图3-1中，Ⅰ、Ⅲ、Ⅴ、Ⅶ、…区域，误差朝绝对值减小的方向变化。此时，可采取保持等待措施，相当于实施开环控制；Ⅱ、Ⅳ、Ⅵ、Ⅷ、…区域，误差绝对值朝增大的方向变化。此时，可根据误差的大小分别实施较强或一般的控制作用，以抑制动态误差。

3.1.2 仿真程序及分析

仿真实例

求三阶传递函数的阶跃响应：

$$G_p(s)=\frac{523500}{s^3+87.35s^2+10470s}$$

式中，采样时间为1ms。

采用专家PID设计控制器。在仿真过程中，ε取0.001，程序中的五条规则与控制算法的五种情况相对应。

仿真方法一

采用M语言进行仿真，仿真结果如图3-2和图3-3所示。

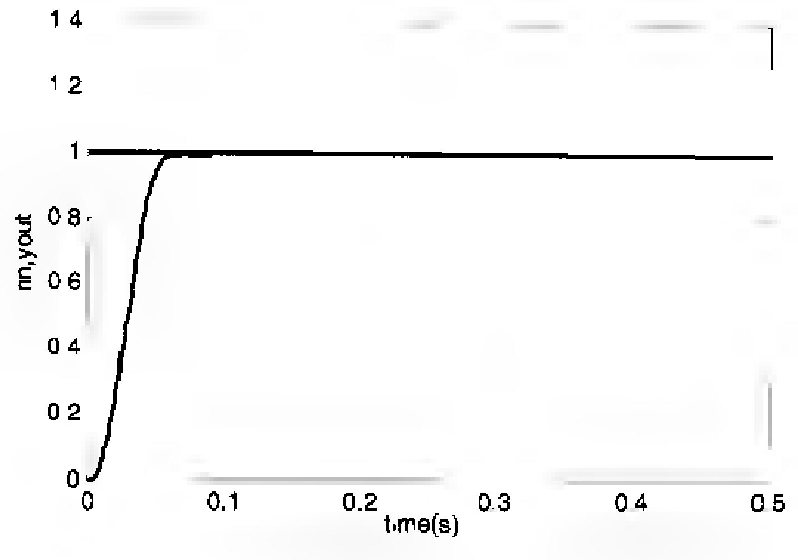


图 3-2　专家 PID 控制阶跃响应曲线

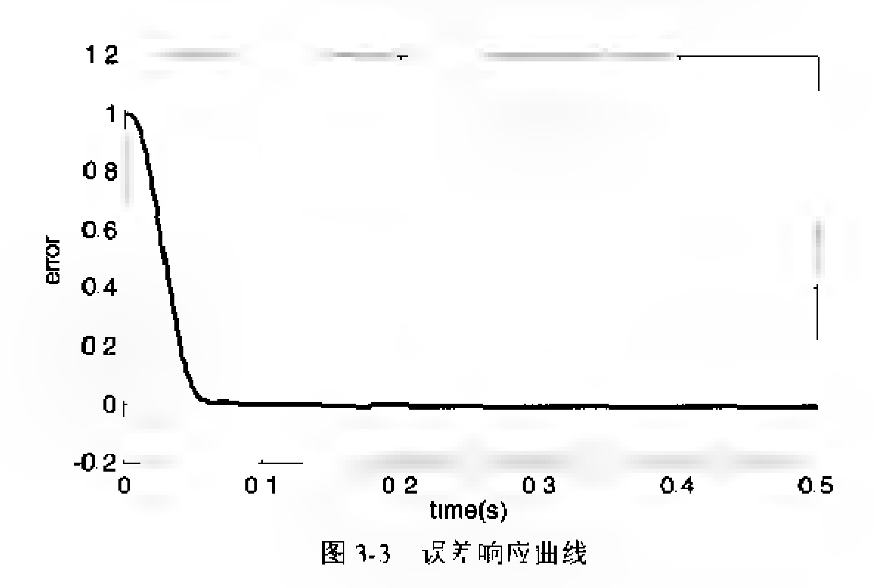


图 3-3　误差响应曲线

仿真程序：chap3_1.m。

```
%Expert PID Controller
clear all;
close all;
ts=0.001;

sys=tf(5.235e005,[1,87.35,1.047e004,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

u_1=0.0;u_2=0.0;u_3=0.0;
y_1=0;y_2=0;y_3=0;

x=[0,0,0]';
```

```
x2_1=0;

kp=0.6;
ki=0.03;
kd=0.01;

error_1=0;
for k=1:1:500
time(k)=k*ts;

rin(k)=1.0;                     %Tracing Jieyue Signal

u(k)=kp*x(1)+kd*x(2)+ki*x(3); %PID Controller

%Expert control rule
if abs(x(1))>0.8      %Rule1:Unclosed control firstly
   u(k)=0.45;
elseif abs(x(1))>0.40
   u(k)=0.40;
elseif abs(x(1))>0.20
   u(k)=0.12;
elseif abs(x(1))>0.01
   u(k)=0.10;
end

if x(1)*x(2)>0|(x(2)==0)       %Rule2
   if abs(x(1))>=0.05
      u(k)=u_1+2*kp*x(1);
   else
      u(k)=u_1+0.4*kp*x(1);
   end
end

if (x(1)*x(2)<0&x(2)*x2_1>0)|(x(1)==0)     %Rule3
    u(k)=u(k);
end

if x(1)*x(2)<0&x(2)*x2_1<0   %Rule4
   if abs(x(1))>=0.05
      u(k)=u_1+2*kp*error_1;
   else
```

```
      u(k)=u_1+0.6*kp*error_1;
   end
end

if abs(x(1))<=0.001   %Rule5:integration separation PI control
   u(k)=0.5*x(1)+0.010*x(3);
end

%Restricting the output of controller
if u(k)>=10
   u(k)=10;
end
if u(k)<=-10
   u(k)=-10;
end

%Linear model
yout(k)=-den(2)*y_1-den(3)*y_2-den(4)*y_3+num(1)*u(k)+num(2)*u_1+
        num(3)*u_2+num(4)*u_3;
error(k)=rin(k)-yout(k);

%------------Return of PID parameters-------------%
u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);

x(1)=error(k);                % Calculating P
x2_1=x(2);
x(2)=(error(k)-error_1)/ts;   % Calculating D
x(3)=x(3)+error(k)*ts;        % Calculating I

error_1=error(k);
end
figure(1);
plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,rin-yout,'r');
xlabel('time(s)');ylabel('error');
```

仿真方法二

采用 Simulink 进行仿真，控制器由一阶 S 函数与 Simulink 模块组合的形式实现。控制器参数、控制输入上下限及采样时间用封装的形式设定，封装界面如图 3-4 所示。仿真结果如图 3-5 所示。

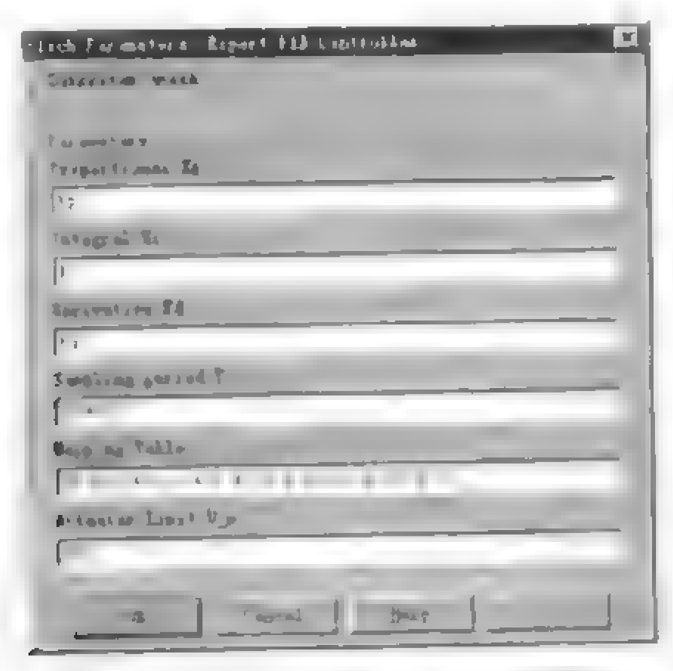


图 3-4 专家 PID 控制的 Simulink 封装界面



图 3-5 专家 PID 控制阶跃响应曲线

仿真程序的 S 函数控制子程序：chap3_2s.m

```
case 3          % computation of control signal
   sys = mdlOutputs(t,x,u,kp,ki,kd,MTab);
case {1, 4, 9}   % unused flag values
   sys = [];
otherwise        % error handling
   error(['Unhandled flag = ',num2str(flag)]);
end;

%--------------------------------------------------------------------------
% when flag=0, perform system initialization
%--------------------------------------------------------------------------
function [sys,x0,str,ts] = mdlInitializeSizes(T)
sizes = simsizes;         % read default control variables
sizes.NumContStates = 0; % no continuous states
sizes.NumDiscStates = 3; % 3 states and assume they are the P/I/D components
sizes.NumOutputs = 2;    % 2 output variables: control u(t) and state x(3)
sizes.NumInputs = 4;     % 4 input signals
sizes.DirFeedthrough = 1;% input reflected directly in output
sizes.NumSampleTimes = 1;% single sampling period
sys = simsizes(sizes);  %
x0 = [0; 0; 0];          % zero initial states
str = [];
ts = [T 0];              % sampling period
%--------------------------------------------------------------------------
% when flag=2, updates the discrete states
%--------------------------------------------------------------------------
function sys = mdlUpdates(x,u,T)
sys = [ u(1);
      x(2)+u(1)*T;
       (u(1)-u(2))/T];
%x(1):error value
%x(2):error integrate
%x(3):error difference
%--------------------------------------------------------------------------
% when flag=3, computates the output signals
%--------------------------------------------------------------------------
function sys = mdlOutputs(t,x,u,kp,ki,kd,MTab)

i=find(abs(x(1))>MTab(:,1));          %Rule1
if length(i)>0
```

```
    sys=MTab(1,2);
else
    sys=[kp,ki,kd]*x;
end

if x(1)*x(3)>0|abs(x(3))<eps          %Rule2
   if abs(x(1))>0.05
      sys=u(3)+2*kp*x(1);
   else
      sys=u(3)+0.4*kp*x(1);
   end
end

if x(1)*x(3)<0 & x(3)*x(4)<0            %Rule4
   if abs(x(1))>0.05
      sys=u(3)+2*kp*x(2);
   else
      sys=u(3)+0.6*kp*x(2);
   end
end

if abs(x(1))<0.1                       %Rule5:integration separation PI control
   sys=0.5*x(1)+0.010*x(2);
end

sys=[sys,x(3)];
```

Simulink 程序：chap3_2.mdl，如图 3-6 和图 3-7 所示。

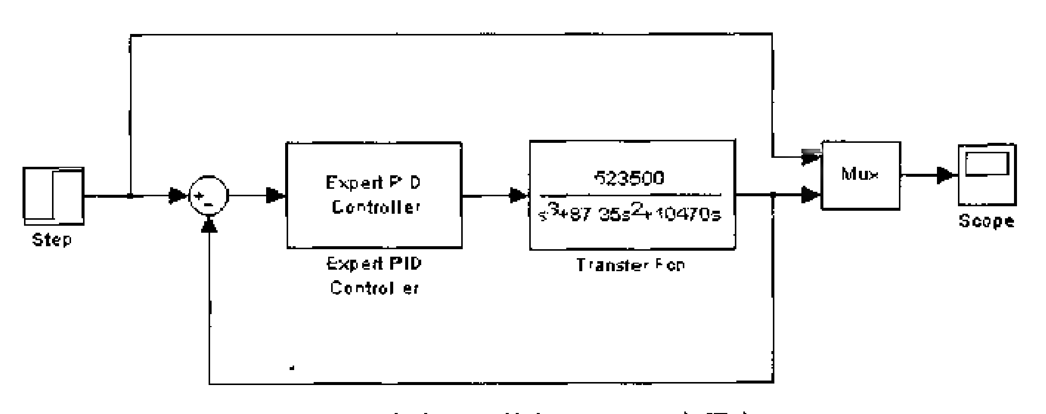


图 3-6　专家 PID 控制 Simulink 主程序

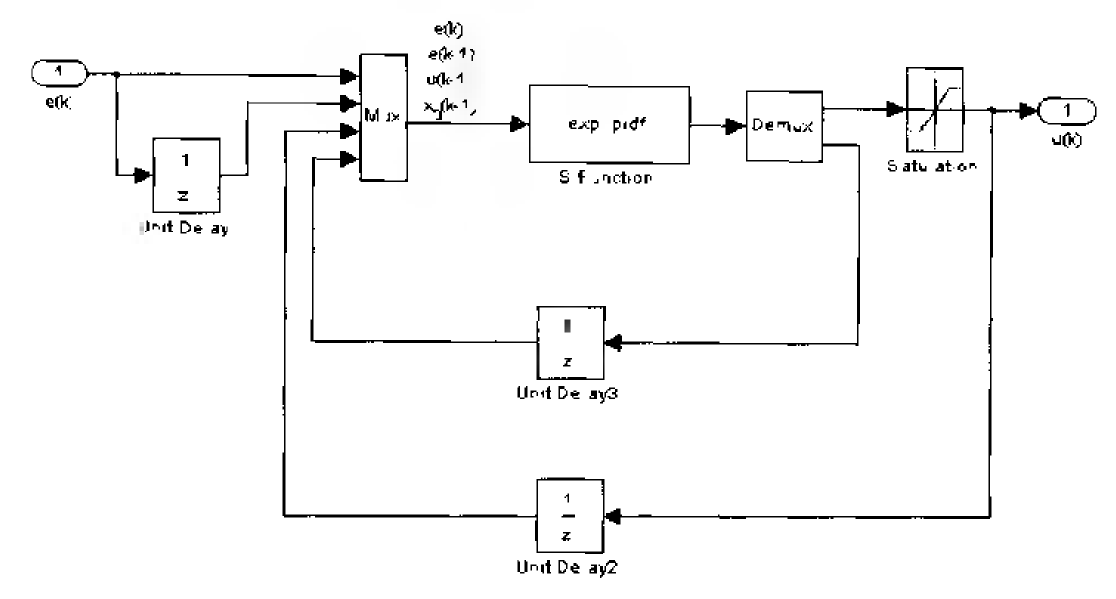


图 3-7 专家PID控制Simulink子程序

3.2 一个典型的模糊控制器的设计

3.2.1 模糊控制的基本原理

模糊控制是以模糊集合论、模糊语言变量及模糊逻辑推理为基础的计算机智能控制，其基本概念是由美国加利福尼亚大学著名教授查德（L. A. Zadeh）首先提出的，经过二十多年的发展，在模糊控制理论和应用研究方面均取得重大成功。

模糊控制的基本原理框图如图 3-8 所示。它的核心部分为模糊控制器，如图中点划线框中所示，模糊控制器的控制规律由计算机的程序实现。实现一步模糊控制算法的过程描述如下：微机经中断采样获取被控制量的精确值，然后将此量与给定值比较得到误差信号 E，一般选误差信号 E 作为模糊控制器的一个输入量。把误差信号 E 的精确量进行模糊化变成模糊量。误差 E 的模糊量可用相应的模糊语言表示，得到误差 E 的模糊语言集合的一个子集 $\underset{\sim}{e}$（$\underset{\sim}{e}$ 是一个模糊矢量），再由 $\underset{\sim}{e}$ 和模糊控制规则 $\underset{\sim}{R}$（模糊算子）根据推理的合成规则进行模糊决策，得到模糊控制量 $\underset{\sim}{u}$：

$$\underset{\sim}{u} = \underset{\sim}{e} \circ \underset{\sim}{R} \tag{3.6}$$

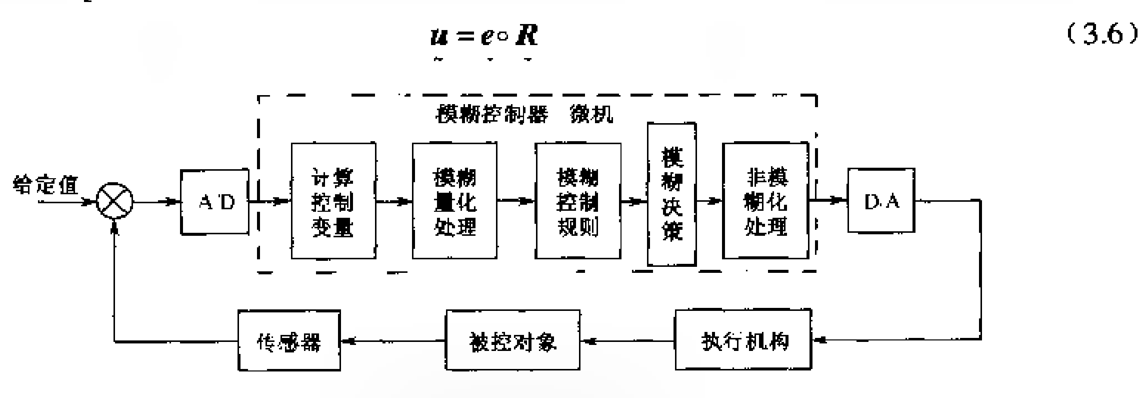


图 3-8 模糊控制的基本原理框图

模糊控制系统与通常的计算机数字控制系统的主要差别是采用了模糊控制器。模糊控制

器是模糊控制系统的核心，一个模糊控制系统的性能优劣主要取决于模糊控制器的结构、所采用的模糊规则、合成推理算法，以及模糊决策的方法等因素。

模糊控制器（Fuzzy Controller，FC）也称为模糊逻辑控制器（Fuzzy Logic Controller，FLC），由于所采用的模糊控制规则是由模糊理论中模糊条件语句描述的，因此模糊控制器是一种语言型控制器，故也称为模糊语言控制器（Fuzzy Language Controller，FLC）。

模糊控制器的组成框图如图 3-9 所示。

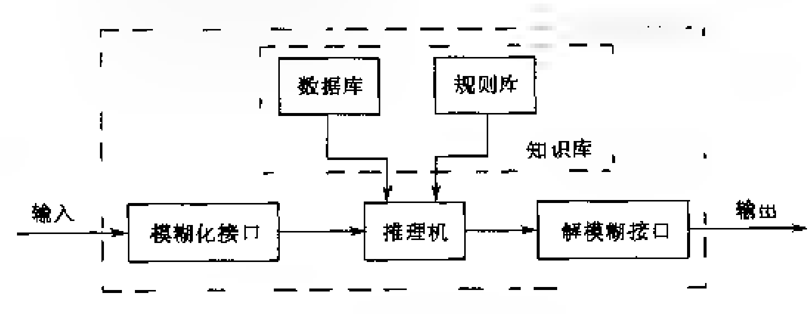


图 3-9　模糊控制器的组成框图

（1）模糊化接口（Fuzzy Interface）

模糊控制器的输入必须通过模糊化才能用于控制输出的求解，因此实际上它是模糊控制器的输入接口。其主要作用是将真实的确定量输入转换为一个模糊矢量。对于一个模糊输入变量 $\underset{\sim}{e}$，其模糊子集通常可以进行如下划分：

$\underset{\sim}{e}$ ={负大，负小，零，正小，正大}={NB, NS, ZO, PS, PB}

$\underset{\sim}{e}$ ={负大，负中，负小，零，正小，正中，正大}={NB, NM, NS, ZO, PS, PM, PB}

$\underset{\sim}{e}$ ={大，负中，负小，零负，零正，正小，正中，正大}={NB, NM, NS, NZ, PZ, PS, PM, PB}

用三角形隶属度函数表示如图 3-10 所示。

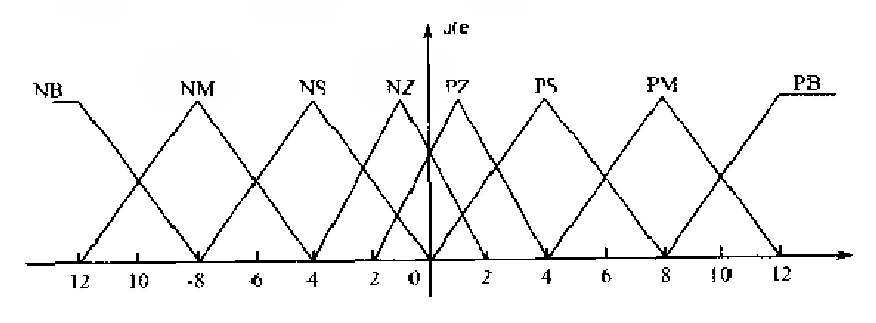


图 3-10　模糊子集和模糊化等级

（2）知识库（Knowledge Base，KB）

知识库由数据库和规则库两部分构成。

a. 数据库（Data Base，DB）　数据库所存放的是所有输入、输出变量的全部模糊子集的隶属度矢量值（经过论域等级离散化以后对应值的集合），若论域为连续域则为隶属度函数。在规则推理的模糊关系方程求解过程中，向推理机提供数据。

b. 规则库（Rule Base，RB）　模糊控制器的规则基于专家知识或手动操作人员长期积累的经验，它是按人的直觉推理的一种语言表示形式。模糊规则通常由一系列的关系词连接而成，如 if then、else、also、end、or 等，关系词必须经过“翻译”才能将模糊规则数值化。

最常用的关系词为 if_then、also，对于多变量模糊控制系统，还有 and 等。例如，某模糊控制系统输入变量为 e（误差）和 ec（误差变化），它们对应的语言变量为 E 和 EC，可给出一组模糊规则：

R1: IF E is NB and EC is NB then U is PB

R2: IF E is NB and EC is NS then U is PM

通常把 if…部分称为"前提部"，而 then…部分称为"结论部"，其基本结构可归纳为 If A and B then C，其中 A 为论域 U 上的一个模糊子集，B 是论域 V 上的一个模糊子集。根据人工控制经验，可离线组织其控制决策表 R，R 是笛卡儿乘积集 $U\times V$ 上的一个模糊子集，则某一时刻其控制量由下式给出：

$$C=(A\times B)\circ R \tag{3.7}$$

式中，× ——模糊直积运算；

∘ ——模糊合成运算。

规则库是用来存放全部模糊控制规则的，在推理时为"推理机"提供控制规则。由上述可知，规则条数和模糊变量的模糊子集划分有关，划分越细，规则条数越多，但并不代表规则库的准确度越高，规则库的"准确性"还与专家知识的准确度有关。

3）推理与解模糊接口（Inference and Defuzzy-interface）

推理是模糊控制器中，根据输入模糊量，由模糊控制规则完成模糊推理来求解模糊关系方程，并获得模糊控制量的功能部分。在模糊控制中，考虑到推理时间，通常采用运算较简单的推理方法。最基本的有 Zadeh 近似推理，它包含有正向推理和逆向推理两类。正向推理常被用于模糊控制中，而逆向推理一般用于知识工程学领域的专家系统中。

推理结果的获得，表示模糊控制的规则推理功能已经完成。但是，至此所获得的结果仍是一个模糊矢量，不能直接用来作为控制量，还必须进行一次转换，求得清晰的控制量输出，即为解模糊。通常把输出端具有转换功能作用的部分称为解模糊接口。

综上所述，模糊控制器实际上是依靠微机（或单片机）构成的。它的绝大部分功能由计算机程序来完成。随着专用模糊芯片的研究和开发，也可以由硬件逐步取代各组成单元的软件功能。

3.2.2 模糊控制器设计步骤

模糊控制器最简单的实现方法是将一系列模糊控制规则离线转化为一个查询表（又称为控制表），存储在计算机中供在线控制时使用。这种模糊控制具结构简单，使用方便，是最基本的一种形式。本节以单变量二维模糊控制器为例，介绍这种形式模糊控制器的设计步骤，其设计思想是设计其他模糊控制器的基础。

（1）模糊控制器的结构

如前所述，单变量二维模糊控制器是最常见的结构形式。

（2）确定模糊控制器的结构

单变量二维模糊控制器是最常见的结构形式。对误差 E、误差变化 EC 及控制量 u 的模糊集及其论域定义如下：

E、EC 和 u 的模糊集均为：{NB, NM, NS, Z, PS, PM, PB}

E、EC 的论域均为：{-3, -2, -1, 0, 1, 2, 3}

u 的论域为：{-4.5，3，1 5, 0, 1, 3, 4.5}

（3）建立模糊控制规则

根据人的直觉思维推理，有系统输出的误差及误差的变化趋势来消除系统误差的模糊控制规则。模糊控制规则语句构成了描述众多被控过程的模糊模型，例如，卫星的姿态与作用的关系，飞机或舰船航向与舵偏角的关系，工业锅炉中的压力与加热的关系等。因此在条件语句中，对于不同的被控对象，误差 E、误差变化 EC 及控制量 u 有不同的意义。

（4）确定模糊变量的赋值表

模糊变量误差 E、误差变化 EC 及控制量 u 的模糊集和论域确定后，须对模糊语言变量确定隶属函数，即所谓对模糊变量赋值，就是确定论域内元素对模糊语言变量的隶属度。

（5）建立模糊控制表

上述描写的模糊控制规则可采用模糊规则表（见表 3-1）来描述，共 49 条模糊规则，各个模糊语句之间是或的关系，由第一条语句所确定的控制规则可以计算出 u_1。同理，可以由其余各条语句分别求出控制量 u_2、…, u_{49}，则控制量为模糊集合 u，可表示为：

$$u=u_1+u_2+\cdots+u_{49} \tag{3.8}$$

表 3-1 模糊规则表

ec \ e (u)	NB	NM	NS	ZO	PS	PM	PB
NB	PB	PB	PM	PM	PS	ZO	ZO
NM	PB	PB	PM	PS	PS	ZO	NS
NS	PM	PM	PM	PS	ZO	NS	NS
ZO	PM	PM	PS	ZO	NS	NM	NM
PS	PS	PS	ZO	NS	NS	NM	NM
PM	PS	ZO	NS	NM	NM	NM	NB
PB	ZO	ZO	NM	NM	NM	NB	NB

（6）去模糊化

由式（3.8）计算出的模糊控制量可以选用一种判决方法，如采用最大隶属度方法，将控制量由模糊量变为精确量。为了获得准确的控制量，就要求模糊方法能够很好地表达输出隶属度函数的计算结果。本文采用工业控制中广泛使用去模糊方法—加权平均法。该法针对论域中的每个元素 x_i ($i=1,2,\cdots,n$)，以它作为待判决输出模糊集合的隶属度$\mu(i)$的加权系数，即取乘积 $x_i\mu(i)$，再计算该乘积和 $\sum_{i=1}^{n}x_i\mu(i)$ 对于隶属度和的平均值 x_0，即：

$$x_0=\frac{\sum_{i=1}^{n}x_i\mu(i)}{\sum_{i=1}^{n}\mu(i)} \tag{3.9}$$

平均值 x_0 便是应用加权平均法为模糊集合求得的判决结果。最后，用输出量化因子乘以

x_0 以适应控制要求，从而可得到控制量的实际值。

3.2.3 模糊控制器设计实例

根据上述步骤，设计二输入单输出模糊控制仿真程序。输入为偏差和偏差变化率，输出为控制信号。通过仿真得到控制器的响应表如下：

fuzzy controller table:e=[-3,+3],ec=[-3,+3]

Ulist =

4	4	2	2	-1	1	0
-4	-2	-2	-1	1	0	2
-2	-2	1	1	0	2	2
2	1	-1	0	2	2	3
-1	-1	0	2	2	3	3
-1	0	2	2	3	3	5
0	2	2	3	3	5	5

运行 plotfis(a2)，得到模糊推理系统，如图 3-11 所示。系统的输入输出隶属度函数如图 3-12～图 3-14 所示。

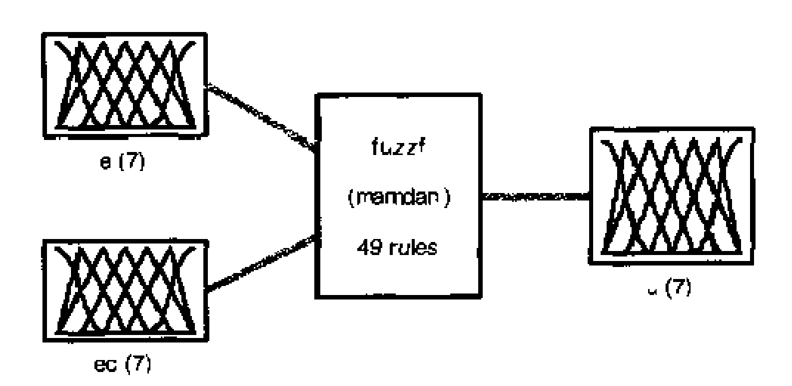


图 3-11 模糊推理系统

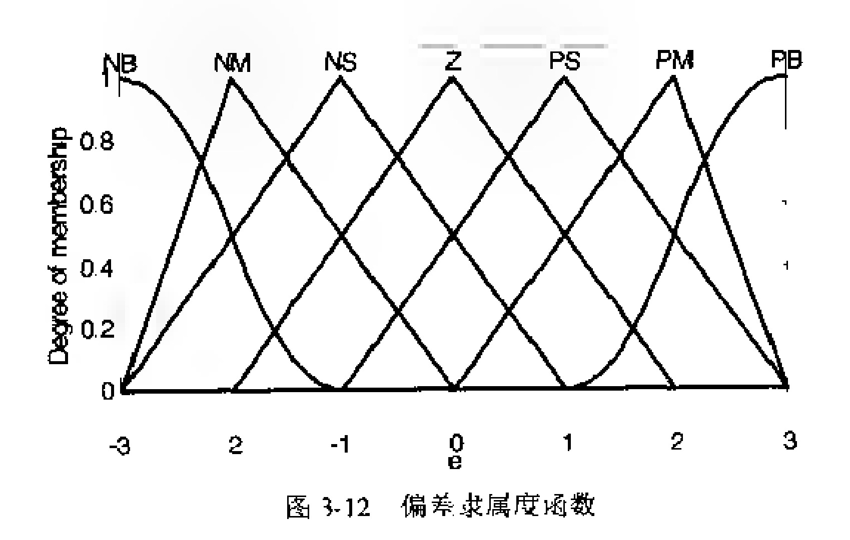


图 3-12 偏差隶属度函数

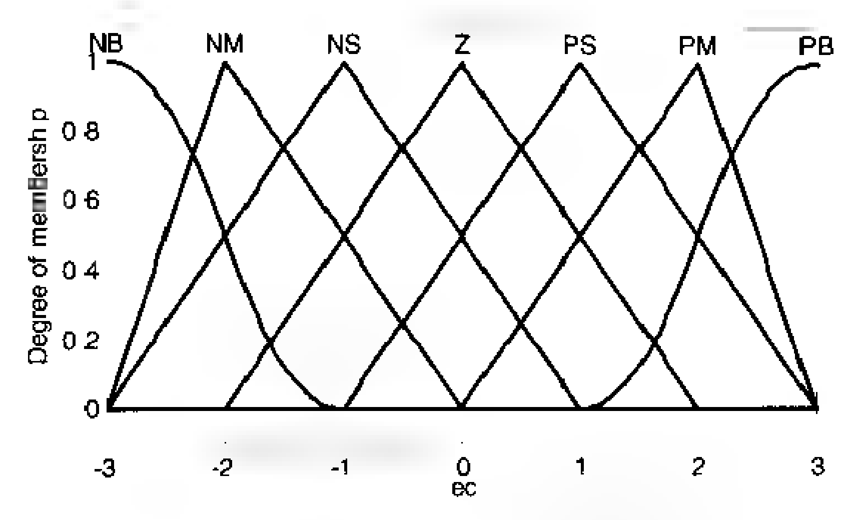


图3-13 偏差变化率隶属度函数

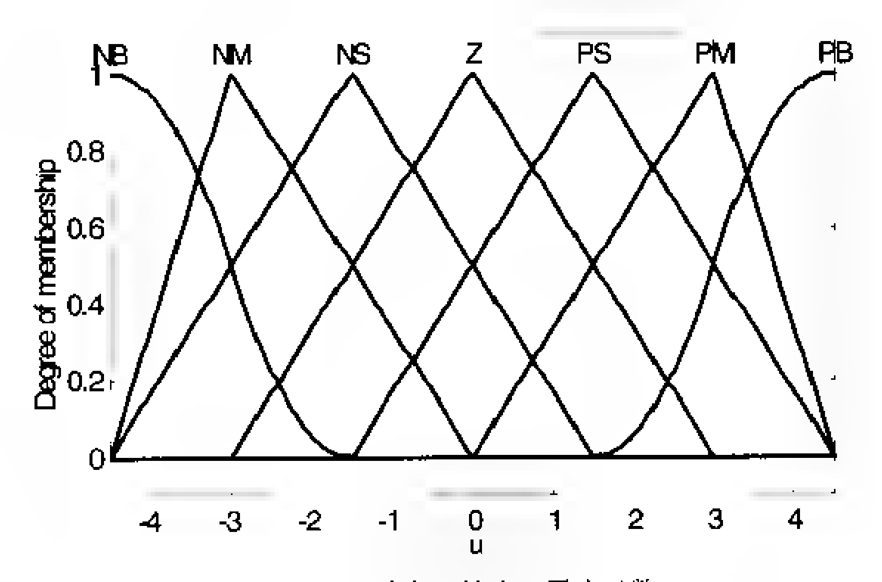


图3-14 控制器输出隶属度函数

仿真程序中模糊控制器的设计：chap3_3.m。

```
%Fuzzy Controller
clear all;
close all;

a=newfis('fuzzf');

f1=1;
a=addvar(a,'input','e',[-3*f1,3*f1]);                   %Parameter e
a=addmf(a,'input',1,'NB','zmf',[-3*f1,-1*f1]);
a=addmf(a,'input',1,'NM','trimf',[-3*f1,-2*f1,0]);
a=addmf(a,'input',1,'NS','trimf',[-3*f1,-1*f1,1*f1]);
a=addmf(a,'input',1,'Z','trimf',[-2*f1,0,2*f1]);
a=addmf(a,'input',1,'PS','trimf',[-1*f1,1*f1,3*f1]);
a=addmf(a,'input',1,'PM','trimf',[0,2*f1,3*f1]);
a=addmf(a,'input',1,'PB','smf',[1*f1,3*f1]);
```

```
f2=1;
a=addvar(a,'input','ec',[-3*f2,3*f2]);          %Parameter ec
a=addmf(a,'input',2,'NB','zmf',[-3*f2,-1*f2]);
a=addmf(a,'input',2,'NM','trimf',[-3*f2,-2*f2,0]);
a=addmf(a,'input',2,'NS','trimf',[-3*f2,-1*f2,1*f2]);
a=addmf(a,'input',2,'Z','trimf',[-2*f2,0,2*f2]);
a=addmf(a,'input',2,'PS','trimf',[-1*f2,1*f2,3*f2]);
a=addmf(a,'input',2,'PM','trimf',[0,2*f2,3*f2]);
a=addmf(a,'input',2,'PB','smf',[1*f2,3*f2]);

f3=1.5;
a=addvar(a,'output','u',[-3*f3,3*f3]);          %Parameter u
a=addmf(a,'output',1,'NB','zmf',[-3*f3,-1*f3]);
a=addmf(a,'output',1,'NM','trimf',[-3*f3,-2*f3,0]);
a=addmf(a,'output',1,'NS','trimf',[-3*f3,-1*f3,1*f3]);
a=addmf(a,'output',1,'Z','trimf',[-2*f3,0,2*f3]);
a=addmf(a,'output',1,'PS','trimf',[-1*f3,1*f3,3*f3]);
a=addmf(a,'output',1,'PM','trimf',[0,2*f3,3*f3]);
a=addmf(a,'output',1,'PB','smf',[1*f3,3*f3]);

rulelist=[1 1 1 1 1;         %Edit rule base
          1 2 1 1 1;
          1 3 2 1 1;
          1 4 2 1 1;
          1 5 3 1 1;
          1 6 3 1 1;
          1 7 4 1 1;

          2 1 1 1 1;
          2 2 2 1 1;
          2 3 2 1 1;
          2 4 3 1 1;
          2 5 3 1 1;
          2 6 4 1 1;
          2   5 1 1;

          3 1 2 1 1;
          3 2 2 1 1;
          3 3 3 1 1;
          3 4 3 1 1;
          3 5 4 1 1;
          3 6 5 1 1;
          3 7 5 1 1;
```

```
4 1 2 1 1;
4 2 3 1 1;
4 3 3 1 1;
4 4 4 1 1;
4 5 5 1 1;
4 6 5 1 1;
4 7 6 1 1;

5 1 3 1 1;
5 2 3 1 1;
5 3 4 1 1;
5 4 5 1 1;
5 5 5 1 1;
5 6 6 1 1;
5 7 6 1 1;

6 1 3 1 1;
6 2 4 1 1;
6 3 5 1 1;
6 4 5 1 1;
6 5 6 1 1;
6 6 6 1 1;
6 7 7 1 1;

7 1 4 1 1;
7 2 5 1 1;
7 3 5 1 1;
7 4 6 1 1;
7 5 6 1 1;
7 6 7 1 1;
7 7 7 1 1];

a=addrule(a,rulelist);
%showrule(a)                          % Show fuzzy rule base

a1=setfis(a,'DefuzzMethod','mom');     % Defuzzy
writefis(a1,'fuzzf');                % save to fuzzy file "fuzzf.fis" which can be
                                     % simulated with fuzzy tool
a2=readfis('fuzzf');

disp('fuzzy controller table:e=[-3,+3],ec=[-3,+3]');
Ulist=zeros(7,7);
```

```
for i=1:7
   for j=1:7
      e(i)=-4+i;
      ec(j)=-4+j;
      Ulist(i,j)=evalfis([e(i),ec(j)],a2);
   end
end

Ulist=ceil(Ulist)

figure(1);
plotfis(a2);
figure(2);
plotmf(a,'input',1);
figure(3);
plotmf(a,'input',2);
figure(4);
plotmf(a,'output',1);
```

3.2.4 模糊控制位置跟踪

仿真方法一：模糊控制位置跟踪的仿真

被控对象为：

$$G(s)=\frac{133}{s^2+25s}$$

采样时间为1ms，采用 z 变换进行离散化，经过 z 变换后的离散化对象为：

$$\mathrm{yout}(k)=-\mathrm{den}(2)\mathrm{yout}(k-1)-\mathrm{den}(3)\mathrm{yout}(k-2)+\mathrm{num}(2)u(k-1)+\mathrm{num}(3)u(k-2)$$

针对离散系统的方波位置响应，运行模糊控制器程序 chap3_4.m，其中反模糊化采用"Centroid"方法。方波响应及控制器输出结果如图3-15和图3-16所示。

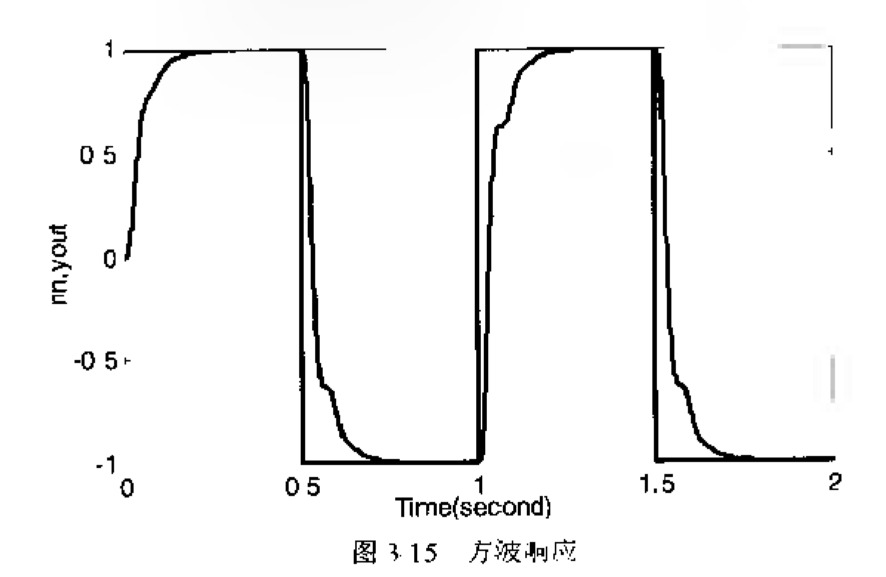


图3-15 方波响应

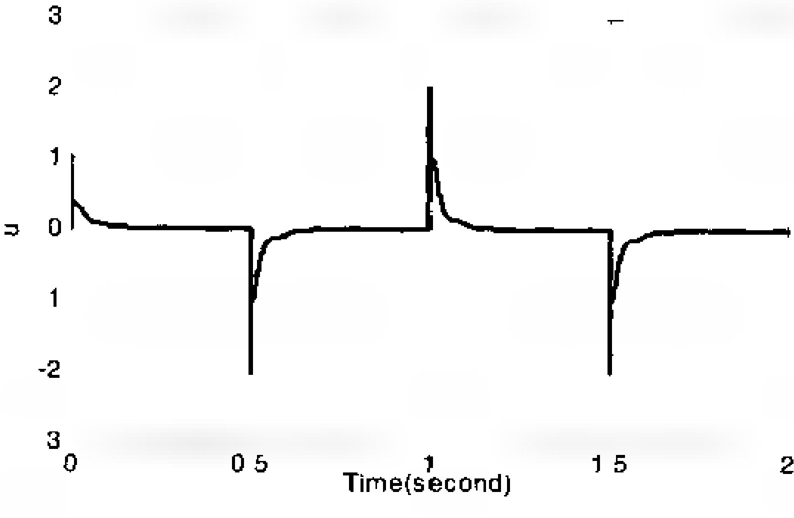


图 3-16　控制器输出

仿真程序：chap3_4.m。

```
%Fuzzy Controller
clear all;close all;

a=newfis('fuzz_ljk');

f1=1.0;
a=addvar(a,'input','e',[-3*f1,3*f1]);                    % Parameter e
a=addmf(a,'input',1,'NB','zmf',[-3*f1,-1*f1]);
a=addmf(a,'input',1,'NM','trimf',[-3*f1,-2*f1,0]);
a=addmf(a,'input',1,'NS','trimf',[-3*f1,-1*f1,1*f1]);
a=addmf(a,'input',1,'Z','trimf',[-2*f1,0,2*f1]);
a=addmf(a,'input',1,'PS','trimf',[-1*f1,1*f1,3*f1]);
a=addmf(a,'input',1,'PM','trimf',[0,2*f1,3*f1]);
a=addmf(a,'input',1,'PB','smf',[1*f1,3*f1]);

f2=1.0;
a=addvar(a,'input','ec',[-3*f2,3*f2]);                    % Parameter ec
a=addmf(a,'input',2,'NB','zmf',[-3*f2,-1*f2]);
a=addmf(a,'input',2,'NM','trimf',[-3*f2,-2*f2,0]);
a=addmf(a,'input',2,'NS','trimf',[-3*f2,-1*f2,1*f2]);
a=addmf(a,'input',2,'Z','trimf',[-2*f2,0,2*f2]);
a=addmf(a,'input',2,'PS','trimf',[-1*f2,1*f2,3*f2]);
a=addmf(a,'input',2,'PM','trimf',[0,2*f2,3*f2]);
a=addmf(a,'input',2,'PB','smf',[1*f2,3*f2]);

%f3=1.4;
f3=1.5;
a=addvar(a,'output','u',[-3*f3,3*f3]);                    % Parameter u
```

```
a=addmf(a,'output',1,'NB','zmf',[-3*f3 -1*f3]);
a=addmf(a,'output',1,'NM','trimf',[-3*f3,-2*f3,0]);
a=addmf(a,'output',1,'NS','trimf',[-3*f3,-1*f3,1*f3]);
a=addmf(a,'output',1,'Z','trimf',[-2*f3,0,2*f3]);
a=addmf(a,'output',1,'PS','trimf',[-1*f3,1*f3,3*f3]);
a=addmf(a,'output',1,'PM','trimf',[0,2*f3,3*f3]);
a=addmf(a,'output',1,'PB','smf',[1*f3,3*f3]);

%Each rule is a PD rule error-rin yout nagative feedback.

rulelist=[1 1 7 1 1;                                   % Edit rule base
          1 2 7 1 1;
          1 3 6 1 1;
          1 4 6 1 1;
          1 5 5 1 1;
          1 6 5 1 1;
          1 7 4 1 1;

          2 1 7 1 1;
          2 2 6 1 1;
          2 3 6 1 1;
          2 4 5 1 1;
          2 5 5 1 1;
          2 6 4 1 1;
          2 7 3 1 1;

          3 1 6 1 1;
          3 2 6 1 1;
          3 3 5 1 1;
          3 4 5 1 1;
          3 5 4 1 1;
          3 6 3 1 1;
          3 7 3 1 1;

          4 1 6 1 1;
          4 2 5 1 1;
          4 3 5 1 1;
          4 4 4 1 1;
          4 5 3 1 1;
          4 6 3 1 1;
          4 7 2 1 1;
```

```
        5 1 5 1 1;
        5 2 5 1 1;
        5 3 4 1 1;
        5 4 3 1 1;
        5 5 3 1 1;
        5 6 2 1 1;
        5 7 2 1 1;

        6 1 5 1 1;
        6 2 5 1 1;
        6 3 4 1 1;
        6 4 3 1 1;
        6 5 2 1 1;
        6 6 2 1 1;
        6 7 1 1 1;

        7 1 4 1 1;
        7 2 3 1 1;
        7 3 3 1 1;
        7 4 2 1 1;
        7 5 1 1 1;
        7 6 1 1 1;
        7 7 1 1 1];

a=addrule(a,rulelist);
%showrule(a)                      % Show fuzzy rule base

a1=setfis(a,'DefuzzMethod','centroid'); % Defuzzy
writefis(a1,'lk');                % save to fuzzy file "lk.fis" which can be
                                  % simulated with fuzzy tool
a2=readfis('lk');
%plotfis(a2);

%fuzzy lk.fis    %Demo fuzzy control simulation
%ruleview(a2);

%%%%%%%%%% Using Fuzzy Controller%%%%%%%%
sys=tf(5.235e005,[1,87.35,1.047e004,0]);
dsys=c2d(sys,0.001,'z');
[num,den]=tfdata(dsys,'v');
u_1=0.0;u_2=0.0;u_3=0.0;
y_1=0;y_2=0;y_3=0;
```

```
error_1=0;
e_1=0.0;
ec_1=0.0;
ei=0;
ts=0.001;
%------------------------------------------
% Start of Control
%------------------------------------------
for k=1:1:2000
time(k)=k*ts;

rin(k)=1*sign(sin(1*2*pi*k*ts));       %Tracing Fangbo Signal

yout(k)=-den(2)*y_1-den(3)*y_2-den(4)*y_3+num(2)*u_1+num(3)*u_2+num(4)*u_3;
error(k)=yout(k)-rin(k);
ei=ei+error(k)*ts;

u(k)=evalfis([e_1 ec_1],a2);     %Using fuzzy inference

u_3=u_2;
u_2=u_1;
u_1=u(k);

y_3=y_2;
y_2=y_1;
y_1=yout(k);

e_1=error(k);
%ec_1=(error(k)-error_1)/ts;
ec_1=error(k)-error_1;

error_2=error_1;
error_1=error(k);
end
figure(1);
plot(time,rin,'b',time,yout,'r');
xlabel('Time(second)');ylabel('rin,yout');
figure(2);
plot(time,u,'r');
xlabel('Time(second)');ylabel('u');
```

仿真方法二　模糊控制位置跟踪的 Simulink 仿真

被控对象为：

$$G(s)=\frac{2500}{s^2+25s}$$

运行模糊控制器程序 chap3_3.m，并将模糊控制器保存在 a2 之中，然后运行模糊控制的 Simulink 程序 chap3_5.mdl，输入取正弦信号，位置跟踪结果如图 3-17 所示。



图 3-17　模糊控制的位置跟踪

仿真程序：chap3_5.mdl，如图 3-18 所示。

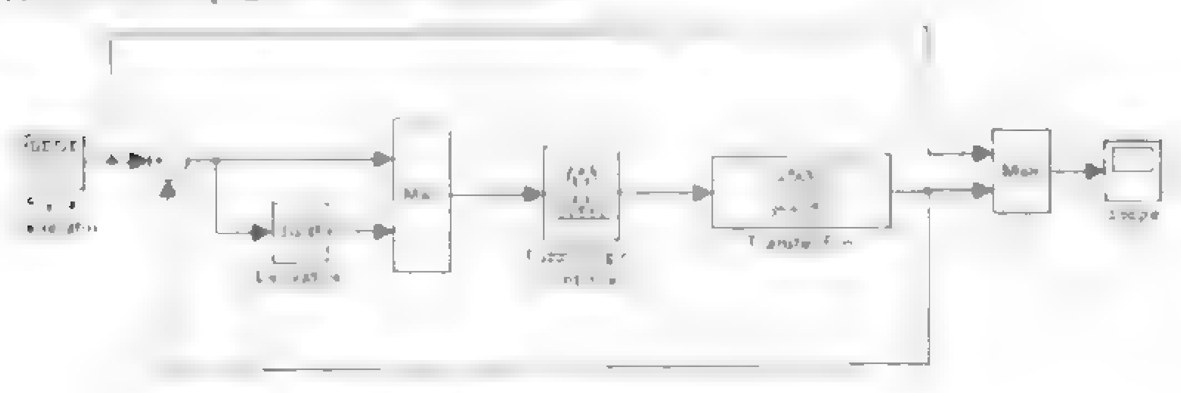


图 3-18　模糊控制的 Simulink 仿真程序

3.3　模糊自适应整定 PID 控制

3.3.1　模糊自适应整定 PID 控制原理

在工业生产过程中，许多被控对象随着负荷变化或干扰因素影响，其对象特性参数或结构发生改变。自适应控制运用现代控制理论在线辨识对象特征参数，实时改变其控制策略，使控制系统品质指标保持在最佳范围内，但其控制效果的好坏取决于辨识模型的精确度，这对于复杂系统是非常困难的。因此，在工业生产过程中，大量采用的仍然是 PID 算法，PID 参数的整定方法很多，但大多数都以对象特性为基础。

随着计算机技术的发展，人们利用人工智能的方法将操作人员的调整经验作为知识存入

计算机中，根据现场实际情况，计算机能自动调整PID参数，这样就出现了智能PID控制器。这种控制器把古典的PID控制与先进的专家系统相结合，实现系统的最佳控制。这种控制必须精确地确定对象模型，首先将操作人员（专家）长期实践积累的经验知识用控制规则模型化，然后运用推理便可对PID参数实现最佳调整。

由于操作者经验不易精确描述，控制过程中各种信号量及评价指标不易定量表示，模糊理论是解决这一问题的有效途径，所以人们运用模糊数学的基本理论和方法，把规则的条件、操作用模糊集表示，并把这些模糊控制规则及有关信息（如评价指标、初始PID参数等）作为知识存入计算机知识库中，然后计算机根据控制系统的实际响应情况（专家系统的输入条件），运用模糊推理，即可自动实现对PID参数的最佳调整，这就是模糊自适应PID控制。目前模糊自适应PID控制器有多种结构形式，但其工作原理基本一致。

自适应模糊PID控制器以误差e和误差变化ec作为输入，可以满足不同时刻的e和ec对PID参数自整定的要求。利用模糊控制规则在线对PID参数进行修改，便构成了自适应模糊PID控制器，其结构如图3-19所示。

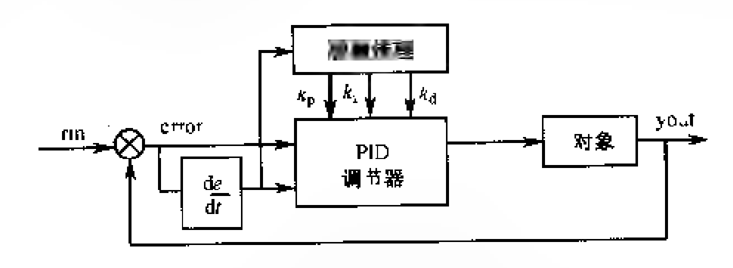


图3-19 自适应模糊控制器结构

PID参数模糊自整定是找出PID三个参数与e和ec之间的模糊关系，在运行中通过不断检测e和ec，根据模糊控制原理来对三个参数进行在线修改，以满足不同e和ec时对控制参数的不同要求，而使被控对象有良好的动、静态性能。

从系统的稳定性、响应速度、超调量和稳态精度等各方面来考虑，k_p, k_i, k_d的作用如下：

（1）比例系数k_p的作用是加快系统的响应速度，提高系统的调节精度。k_p越大，系统的响应速度越快，系统的调节精度越高，但易产生超调，甚至会导致系统不稳定。k_p取值过小，则会降低调节精度，使响应速度缓慢，从而延长调节时间，使系统静态、动态特性变坏。

（2）积分作用系数k_i的作用是消除系统的稳态误差。k_i越大，系统的静态误差消除越快，但k_i过大，在响应过程的初期会产生积分饱和现象，从而引起响应过程的较大超调。若k_i过小，将使系统静态误差难以消除，影响系统的调节精度。

（3）微分作用系数k_d的作用是改善系统的动态特性，其作用主要是在响应过程中抑制偏差向任何方向的变化，对偏差变化进行提前预报。但k_d过大，会使响应过程提前制动，从而延长调节时间，而且会降低系统的抗干扰性能。

PID参数的整定必须考虑到在不同时刻三个参数的作用及相互之间的关系。

在线实时模糊自整定PID控制器控制方案原理如图3-4所示。

模糊自整定PID是在PID算法的基础上，通过计算当前系统误差e和误差变化率ec，利用模糊规则进行模糊推理，查询模糊矩阵表进行参数调整。

模糊控制设计的核心是总结工程设计人员的技术知识和实际操作经验，建立合适的模糊规则表，得到针对k_p，k_i，k_d三个参数分别整定的模糊控制表。

(1) k_p 的模糊规则表见表 3-2。

表 3-2　k_p 的模糊规则表

Δkp	NB	NM	NS	ZO	PS	PM	PB
NB	PB	PB	PM	PM	PS	ZO	ZO
NM	PB	PB	PM	PS	PS	ZO	NS
NS	PM	PM	PM	PS	ZO	NS	NS
ZO	PM	PM	PS	ZO	NS	NM	NM
PS	PS	PS	ZO	NS	NS	NM	NM
PM	PS	ZO	NS	NM	NM	NM	NB
PB	ZO	ZO	NM	NM	NM	NB	NB

(2) k_i 的模糊规则表见表 3-3。

表 3-3　k_i 的模糊规则表

ec Δk_i	NB	NM	NS	ZO	PS	PM	PB
NB	NB	NB	NM	NM	NS	ZO	ZO
NM	NB	NB	NM	NS	NS	ZO	ZO
NS	NB	NM	NS	NS	ZO	PS	PS
ZO	NM	NM	NS	ZO	PS	PM	PM
PS	NM	NS	ZO	PS	PS	PM	PB
PM	ZO	ZO	PS	PS	PM	PB	PB
PB	ZO	ZO	PS	PM	PM	PB	PB

(3) k_d 的模糊控制规则表见表 3-4。

表 3-4　k_d 的模糊控制规则表

ec Δk_d e	NB	NM	NS	ZO	PS	PM	PB
NB	PS	NS	NB	NB	NB	NM	PS
NM	PS	NS	NB	NM	NM	NS	ZO
NS	ZO	NS	NM	NM	NS	NS	ZO
ZO	ZO	NS	NS	NS	NS	NS	ZO
PS	ZO	ZO	ZO	ZO	ZO	ZO	ZO
PM	PB	NS	PS	PS	PS	PS	PB
PB	PB	PM	PM	PM	PS	PS	PB

k_p，k_i，k_d 的模糊控制规则表建立好后，可根据如下方法进行 k_p，k_i，k_d 的自适应校正。

将系统误差 e 和误差变化率 ec 变化范围定义为模糊集上的论域。

$$e, ec = \{-5,-4,-3,-2,-1,0,1,2,3,4,5\} \tag{3.10}$$

其模糊子集为 e, ec = {NB, NM, NS, O, PS, PM, PB}，子集中元素分别代表负大，负中，负小，零，正小，正中，正大。设 e, ec 和 k_p，k_i，k_d 均服从正态分布，因此可得出各模糊子集的隶属度，根据各模糊子集的隶属度赋值表和各参数模糊控制模型，应用模糊合成推理设计 PID 参数的模糊矩阵表，查出修正参数代入下式计算：

$$\begin{aligned} k_p &= k_p' + \{e_i, ec_i\}_p \\ k_i &= k_i' + \{e_i, ec_i\}_i \\ k_d &= k_d' + \{e_i, ec_i\}_d \end{aligned} \tag{3.11}$$

在线运行过程中，控制系统通过对模糊逻辑规则的结果处理、查表和运算，完成对 PID 参数的在线自校正。其工作流程图如图 3-20 所示。

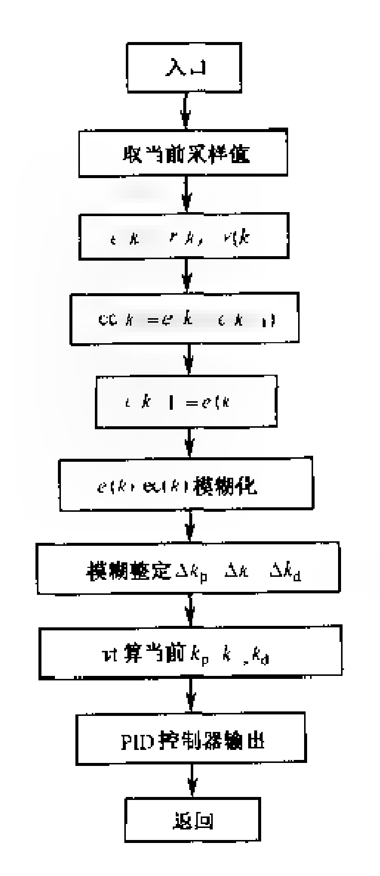


图 3-20　在线自校正工作流程图

3.3.2　仿真程序及分析

仿真实例

被控对象为：

$$G_p(s) = \frac{523500}{s^3 + 87.35s^2 + 10470s}$$

采样时间为 1ms，采用模糊 PID 控制进行阶跃响应，在第 300 个采样时间时控制器输出加 1.0 的干扰，相应的响应结果如图 3-21～图 3-26 所示。

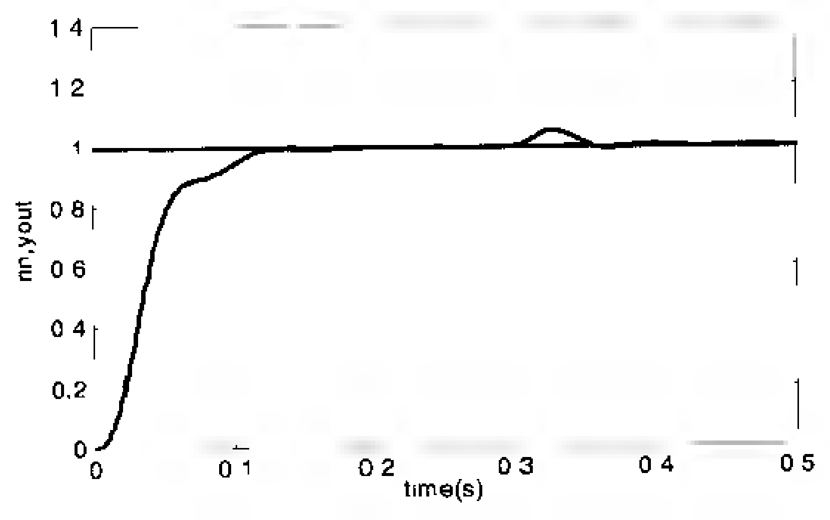


图 3-21　模糊 PID 控制阶跃响应

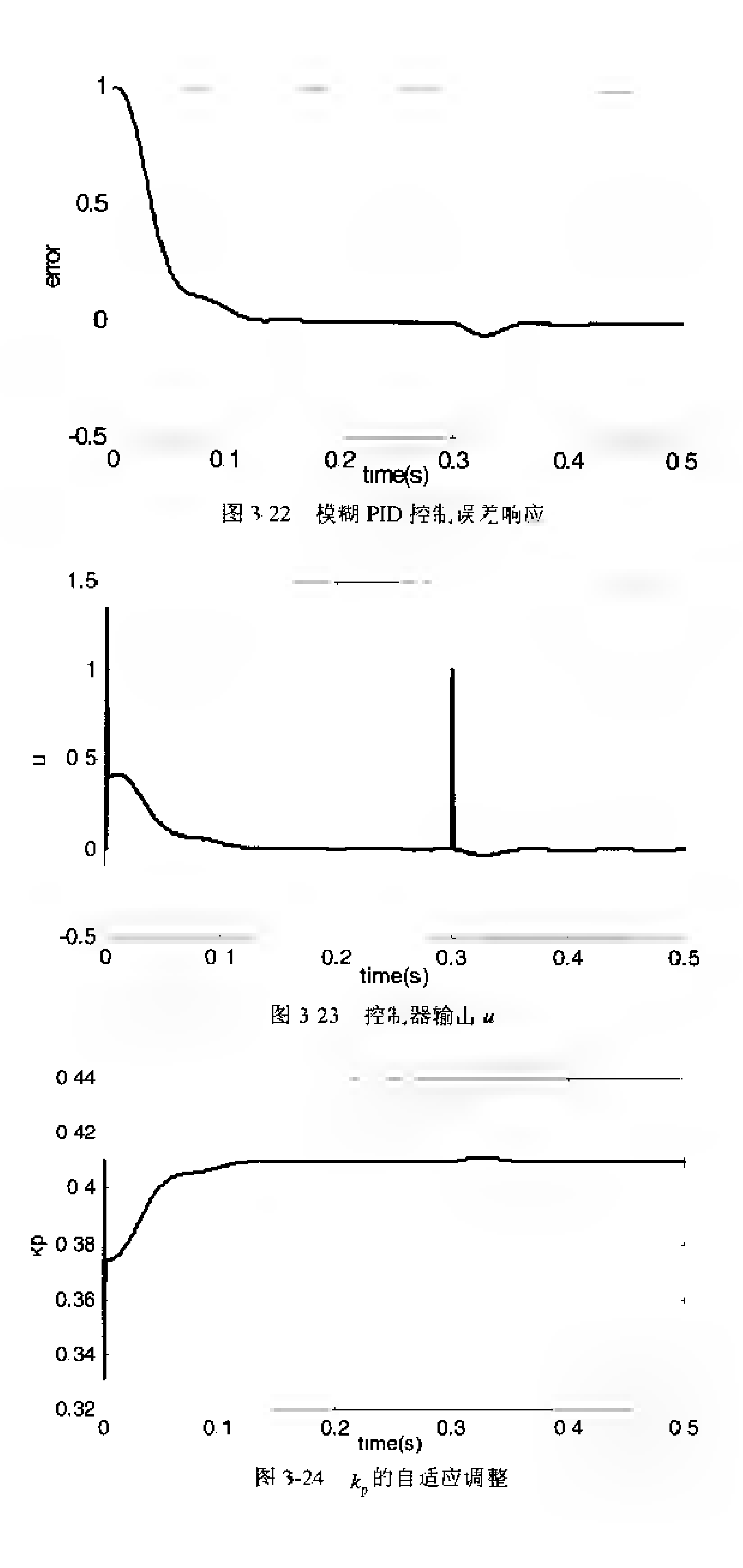


图 3-22 模糊 PID 控制误差响应

图 3-23 控制器输出 u

图 3-24 k_p 的自适应调整

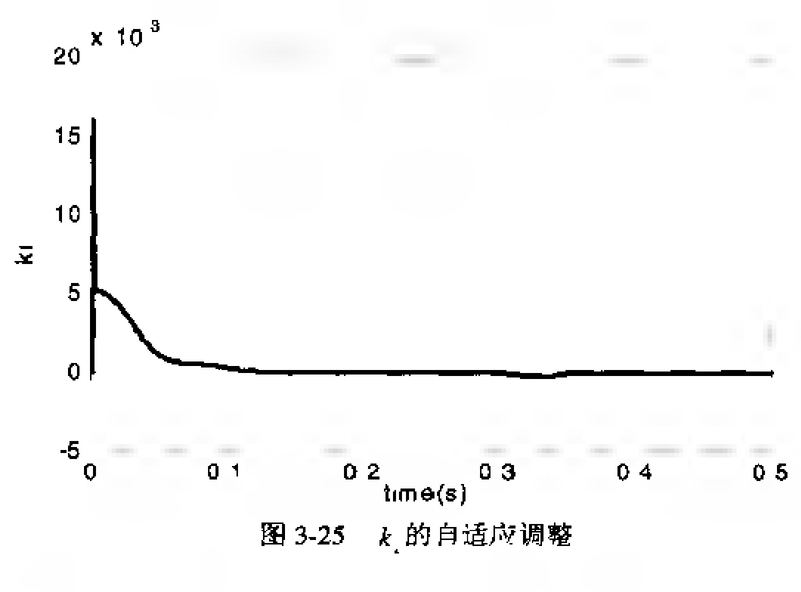


图 3-25　k_i 的自适应调整

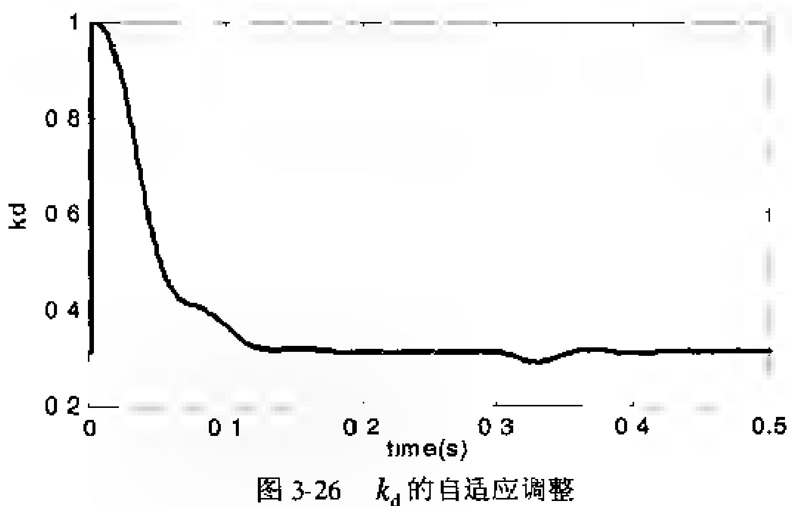


图 3-26　k_d 的自适应调整

仿真程序：chap3_6.m。

```
%Fuzzy Tunning PID Control
clear all;
close all;

a=newfis('fuzzpid');

a=addvar(a,'input','e',[-3,3]);                    %Parameter e
a=addmf(a,'input',1,'NB','zmf',[-3,-1]);
a=addmf(a,'input',1,'NM','trimf',[-3,-2,0]);
a=addmf(a,'input',1,'NS','trimf',[-3,-1,1]);
a=addmf(a,'input',1,'Z','trimf',[-2,0,2]);
a=addmf(a,'input',1,'PS','trimf',[-1,1,3]);
a=addmf(a,'input',1,'PM','trimf',[0,2,3]);
a=addmf(a,'input',1,'PB','smf',[1,3]);

a=addvar(a,'input','ec',[-3,3]);                   %Parameter ec
a=addmf(a,'input',2,'NB','zmf',[-3,-1]);
```

```
a=addmf(a,'input',2,'NM','trimf',[-3,-2,0]);
a=addmf(a,'input',2,'NS','trimf',[-3,-1,1]);
a=addmf(a,'input',2,'Z','trimf',[-2,0,2]);
a=addmf(a,'input',2,'PS','trimf',[-1,1,3]);
a=addmf(a,'input',2,'PM','trimf',[0,2,3]);
a=addmf(a,'input',2,'PB','smf',[1,3]);

a=addvar(a,'output','kp',[-0.3,0.3]);                %Parameter kp
a=addmf(a,'output',1,'NB','zmf',[-0.3,-0.1]);
a=addmf(a,'output',1,'NM','trimf',[-0.3,-0.2,0]);
a=addmf(a,'output',1,'NS','trimf',[-0.3,-0.1,0.1]);
a=addmf(a,'output',1,'Z','trimf',[-0.2,0,0.2]);
a=addmf(a,'output',1,'PS','trimf',[-0.1,0.1,0.3]);
a=addmf(a,'output',1,'PM','trimf',[0,0.2,0.3]);
a=addmf(a,'output',1,'PB','smf',[0.1,0.3]);

a=addvar(a,'output','ki',[-0.06,0.06]);          %Parameter ki
a=addmf(a,'output',2,'NB','zmf',[-0.06,-0.02]);
a=addmf(a,'output',2,'NM','trimf',[-0.06,-0.04,0]);
a=addmf(a,'output',2,'NS','trimf',[-0.06,-0.02,0.02]);
a=addmf(a,'output',2,'Z','trimf',[-0.04,0,0.04]);
a=addmf(a,'output',2,'PS','trimf',[-0.02,0.02,0.06]);
a=addmf(a,'output',2,'PM','trimf',[0,0.04,0.06]);
a=addmf(a,'output',2,'PB','smf',[0.02,0.06]);

a=addvar(a,'output','kd',[-3,3]);              %Parameter kd
a=addmf(a,'output',3,'NB','zmf',[-3,-1]);
a=addmf(a,'output',3,'NM','trimf',[-3,-2,0]);
a=addmf(a,'output',3,'NS','trimf',[-3,-1,1]);
a=addmf(a,'output',3,'Z','trimf',[-2,0,2]);
a=addmf(a,'output',3,'PS','trimf',[-1,1,3]);
a=addmf(a,'output',3,'PM','trimf',[0,2,3]);
a=addmf(a,'output',3,'PB','smf',[1,3]);

rulelist=[1 1 7 1 5 1 1;
          1 2 7 1 3 1 1;
          1 3 6 2 1 1 1;
          1 4 6 2 1 1 1;
          1 5 5 3 1 1 1;
          1 6 4 4 2 1 1;
          1 7 4 4 5 1 1;
```

2 1 7 1 5 1 1,
2 2 7 1 3 1 1;
2 3 6 2 1 1 1;
2 4 5 3 2 1 1;
2 5 5 3 2 1 1,
2 6 4 4 3 1 1;
2 [illegible] 3 4 4 [illegible] 1;

3 1 6 1 4 1 1;
3 2 6 2 3 1 1;
3 3 6 3 2 1 1;
3 4 5 3 2 1 1;
3 5 4 4 3 1 1;
3 6 3 5 3 1 1,
3 7 3 5 4 1 1;

4 1 6 2 4 1 1;
4 2 6 2 3 1 1;
4 3 5 3 3 1 1;
4 4 4 4 3 1 1,
4 5 3 5 3 1 1;
4 6 2 6 3 1 1;
4 7 2 6 4 1 1;

5 1 5 2 4 1 1;
5 2 5 3 4 1 1;
5 3 4 4 4 1 1;
5 4 3 5 4 1 1;
5 5 3 5 4 1 1;
5 6 2 6 4 1 1;
5 7 2 7 4 1 1;

6 1 5 4 7 1 1;
6 2 4 4 5 1 1;
6 3 3 5 5 1 1;
6 4 2 5 5 1 1,
6 5 2 6 5 1 1;
6 6 2 7 5 1 1;
6 7 1 7 7 1 1;

[illegible] 1 4 4 7 1 1,
7 2 4 4 6 1 1;

```
        7 3 2 5 6 1 1;
        7 4 2 6 6 1 1;
        7 5 2 6 5 1 1;
        7 6 1 7 5 1 1;
        7 7 1 7 7 1 1];

a=addrule(a,rulelist);
a=setfis(a,'DefuzzMethod','centroid');
writefis(a,'fuzzpid');

a=readfis('fuzzpid');

%PID Controller
ts=0.001;
sys=tf(5.235e005,[1,87.35,1.047e004,0]);
dsys=c2d(sys,ts,'tustin');
[num,den]=tfdata(dsys,'v');

u_1=0.0;u_2=0.0;u_3=0.0;

y_1=0;y_2=0;y_3=0;

x=[0,0,0]';

error_1=0;
e_1=0.0;
ec_1=0.0;

kp0=0.40;
kd0=1.0;
ki0=0.0;

for k=1:1:500
time(k)=k*ts;

rin(k)=1;
%Using fuzzy inference to tunning PID
k_pid=evalfis([e_1,ec_1],a);
kp(k)=kp0+k_pid(1);
ki(k)=ki0+k_pid(2);
kd(k)=kd0+k_pid(3);
u(k)=kp(k)*x(1)+kd(k)*x(2)+ki(k)*x(3);
```

```
if k==300     % Adding disturbance(1.0v at time 0.3s)
  u(k)=u(k)+1.0;
end
if u(k)>=10
  u(k)=10;
end
if u(k)<=-10
  u(k)=-10;
end

yout(k)=-den(2)*y_1-den(3)*y_2-den(4)*y_3+num(1)*u(k)+num(2)*u_1+
       num(3)*u_2+num(4)*u_3;
error(k)=rin(k)-yout(k);
%%%%%%%%%%%%%Return of PID parameters%%%%%%%%%%%%%
  u_3=u_2;
  u_2=u_1;
  u_1=u(k);

  y_3=y_2;
  y_2=y_1;
  y_1=yout(k);

  x(1)=error(k);                % Calculating P
  x(2)=error(k)-error_1;         % Calculating D
  x(3)=x(3)+error(k);           % Calculating I

  e_1=x(1);
  ec_1=x(2);

  error_2=error_1;
  error_1=error(k);
end
showrule(a)
figure(1);plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('rin,yout');
figure(2);plot(time,error,'r');
xlabel('time(s)');ylabel('error');
figure(3);plot(time,u,'r');
xlabel('time(s)');ylabel('u');
figure(4);plot(time,kp,'r');
xlabel('time(s)');ylabel('kp');
```

```
figure(5);plot(time,ki,'r');
xlabel('time(s)');ylabel('ki');
figure(6);plot(time,kd,'r');
xlabel('time(s)');ylabel('kd');
figure(7);plotmf(a,'input',1);
figure(8);plotmf(a,'input',2);
figure(9);plotmf(a,'output',1);
figure(10);plotmf(a,'output',2);
figure(11);plotmf(a,'output',3);
plotfis(a);
fuzzy fuzzpid
```

通过仿真，还可以得到以下几个结果：

（1）在 MATLAB 下运行 plotmf(a,'input', 1)可得到模糊系统第一个输入 e 的隶属函数，同理可得到 de，k_p，k，k_d 的隶属函数，如图 3-27～图 3-31 所示。

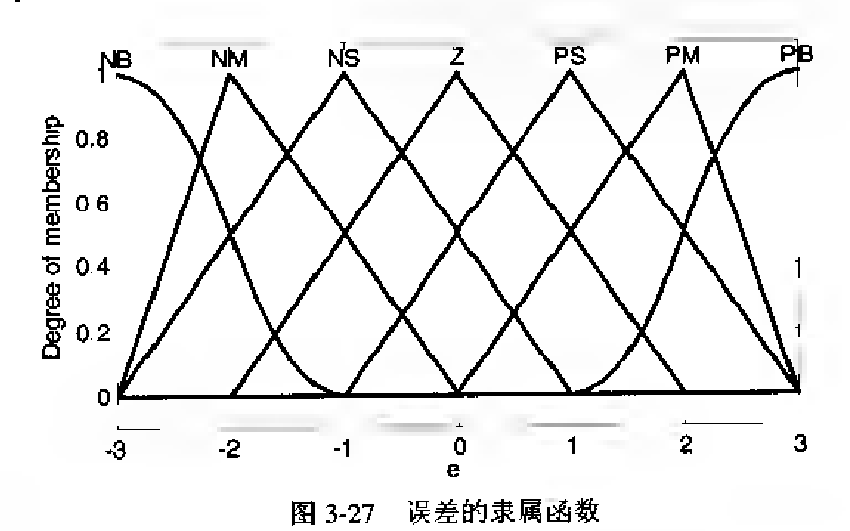


图 3-27　误差的隶属函数

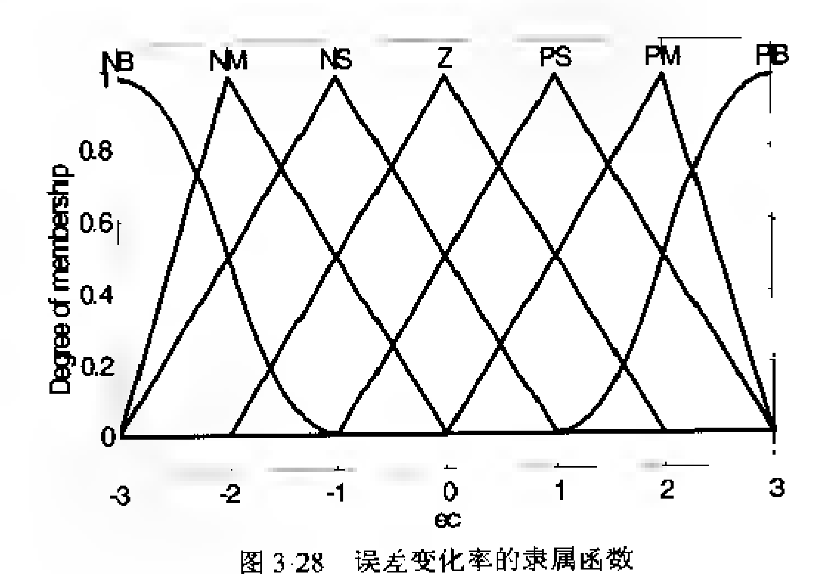


图 3-28　误差变化率的隶属函数

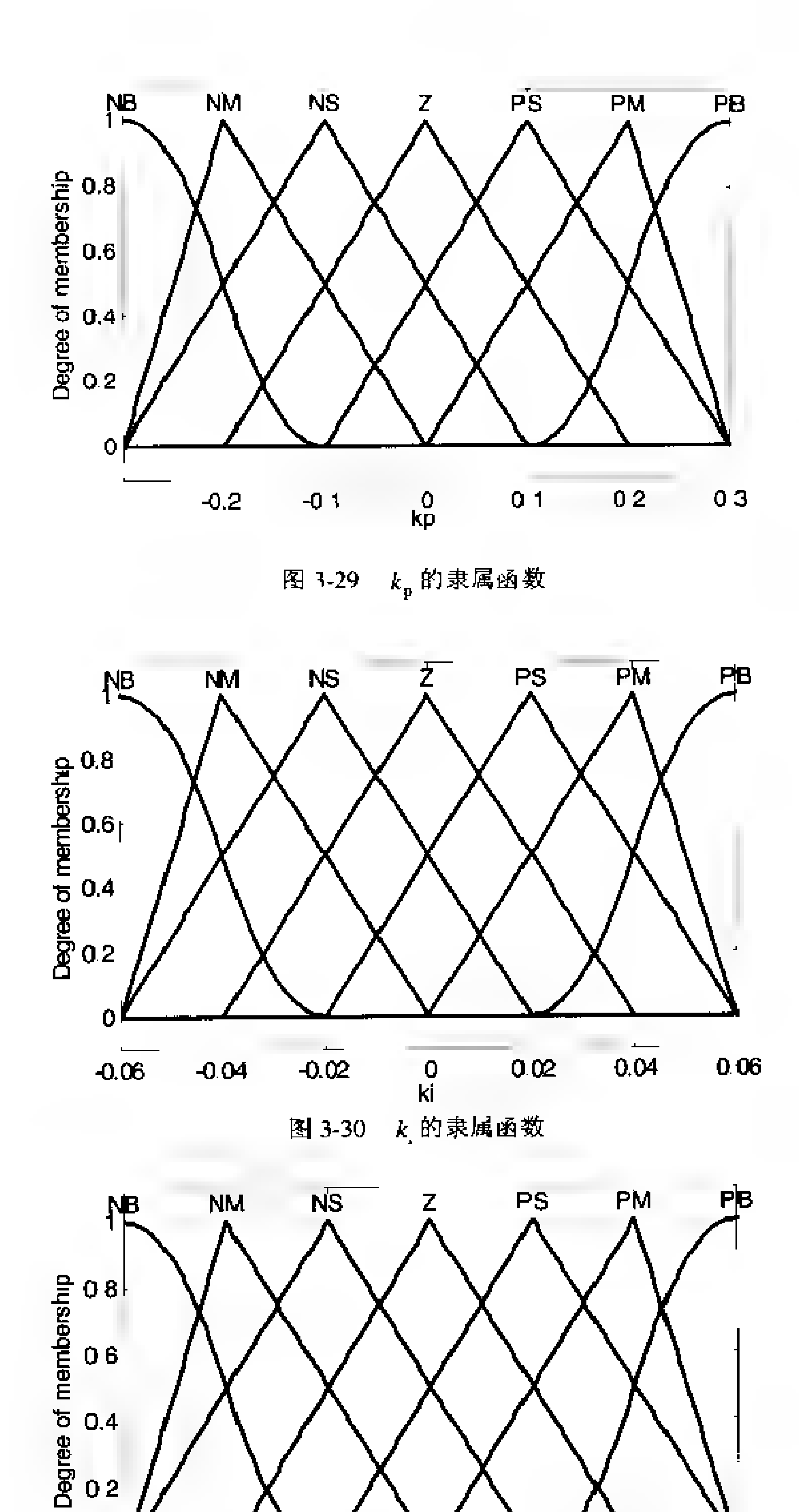


图 3-29　k_p 的隶属函数

图 3-30　k_i 的隶属函数

图 3-31　k_d 的隶属函数

（2）在 MATLAB 下运行 plotfis(a)可观察模糊控制系统的构成，如图 3-32 所示。

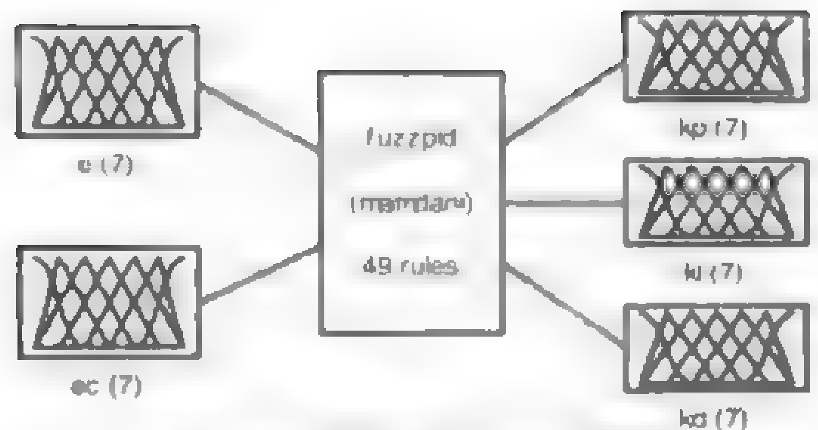


图 3-32 模糊 PID 控制系统构成

3. 在 MATLAB 下运行 fuzzy fuzzpid.fis 可进入 MATLAB 动态仿真工具箱动态仿真环境，如图 3-33 所示。

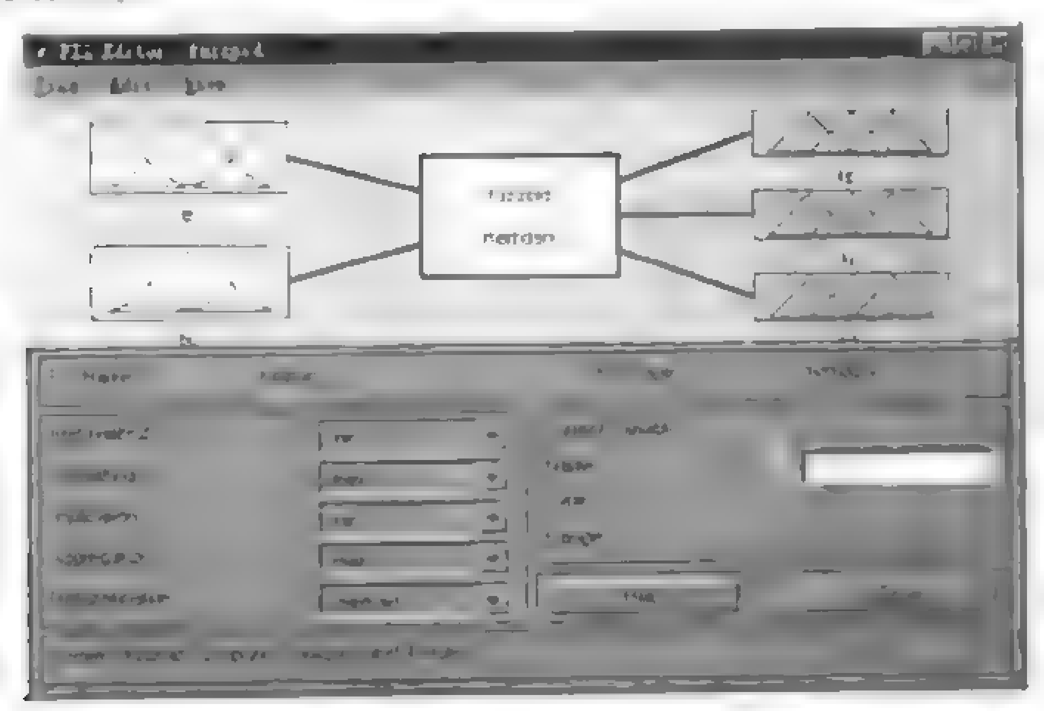


图 3-33 模糊 PID 动态仿真环境

4. MATLAB 下运行 showrule(a)，可得到以下 49 条模糊规则：

1. If (e is NB) and (ec is NB) then (kp is PB)(ki is NB)(kd is PS) (1)
2. If (e is NB) and (ec is NM) then (kp is PB)(ki is NB)(kd is NS) (1)
3. If (e is NB) and (ec is NS) then (kp is PM)(ki is NM)(kd is NB) (1)
4. If (e is NB) and (ec is Z) then (kp is PM)(ki is NM)(kd is NB) (1)
5. If (e is NB) and (ec is PS) then (kp is PS)(ki is NS)(kd is NB) (1)
6. If (e is NB) and (ec is PM) then (kp is Z)(ki is Z)(kd is NM) (1)
7. If (e is NB) and (ec is PB) then (kp is Z)(ki is Z)(kd is PS) (1)
8. If (e is NM) and (ec is NB) then (kp is PB)(ki is NB)(kd is PS) (1)
9. If (e is NM) and (ec is NM) then (kp is PB)(ki is NB)(kd is NS) (1)

10. If (e is NM) and (ec is NS) then (kp is PM)(ki is NM)(kd is NB) (1)
11. If (e is NM) and (ec is Z) then (kp is PS)(ki is NS)(kd is NM) (1)
12. If (e is NM) and (ec is PS) then (kp is PS)(ki is NS)(kd is NM) (1)
13. If (e is NM) and (ec is PM) then (kp is Z)(ki is Z)(kd is NS) (1)
14. If (e is NM) and (ec is PB) then (kp is NS)(ki is Z)(kd is Z) (1)
15. If (e is NS) and (ec is NB) then (kp is PM)(ki is NB)(kd is Z) (1)
16. If (e is NS) and (ec is NM) then (kp is PM)(ki is NM)(kd is NS) (1)
17. If (e is NS) and (ec is NS) then (kp is PM)(ki is NS)(kd is NM) (1)
18. If (e is NS) and (ec is Z) then (kp is PS)(ki is NS)(kd is NM) (1)
19. If (e is NS) and (ec is PS) then (kp is Z)(ki is Z)(kd is NS) (1)
20. If (e is NS) and (ec is PM) then (kp is NS)(ki is PS)(kd is NS) (1)
21. If (e is NS) and (ec is PB) then (kp is NS)(ki is PS)(kd is Z) (1)
22. If (e is Z) and (ec is NB) then (kp is PM)(ki is NM)(kd is Z) (1)
23. If (e is Z) and (ec is NM) then (kp is PM)(ki is NM)(kd is NS) (1)
24. If (e is Z) and (ec is NS) then (kp is PS)(ki is NS)(kd is NS) (1)
25. If (e is Z) and (ec is Z) then (kp is Z)(ki is Z)(kd is NS) (1)
26. If (e is Z) and (ec is PS) then (kp is NS)(ki is PS)(kd is NS) (1)
27. If (e is Z) and (ec is PM) then (kp is NM)(ki is PM)(kd is NS) (1)
28. If (e is Z) and (ec is PB) then (kp is NM)(ki is PM)(kd is Z) (1)
29. If (e is PS) and (ec is NB) then (kp is PS)(ki is NM)(kd is Z) (1)
30. If (e is PS) and (ec is NM) then (kp is PS)(ki is NS)(kd is Z) (1)
31. If (e is PS) and (ec is NS) then (kp is Z)(ki is Z)(kd is Z) (1)
32. If (e is PS) and (ec is Z) then (kp is NS)(ki is PS)(kd is Z) (1)
33. If (e is PS) and (ec is PS) then (kp is NS)(ki is PS)(kd is Z) (1)
34. If (e is PS) and (ec is PM) then (kp is NM)(ki is PM)(kd is Z) (1)
35. If (e is PS) and (ec is PB) then (kp is NM)(ki is PB)(kd is Z) (1)
36. If (e is PM) and (ec is NB) then (kp is PS)(ki is Z)(kd is PB) (1)
37. If (e is PM) and (ec is NM) then (kp is Z)(ki is Z)(kd is PS) (1)
38. If (e is PM) and (ec is NS) then (kp is NS)(ki is PS)(kd is PS) (1)
39. If (e is PM) and (ec is Z) then (kp is NM)(ki is PS)(kd is PS) (1)
40. If (e is PM) and (ec is PS) then (kp is NM)(ki is PM)(kd is PS) (1)
41. If (e is PM) and (ec is PM) then (kp is NM)(ki is PB)(kd is PS) (1)
42. If (e is PM) and (ec is PB) then (kp is NB)(ki is PB)(kd is PB) (1)
43. If (e is PB) and (ec is NB) then (kp is Z)(ki is Z)(kd is PB) (1)
44. If (e is PB) and (ec is NM) then (kp is Z)(ki is Z)(kd is PM) (1)
45. If (e is PB) and (ec is NS) then (kp is NM)(ki is PS)(kd is PM) (1)
46. If (e is PB) and (ec is Z) then (kp is NM)(ki is PM)(kd is PM) (1)
47. If (e is PB) and (ec is PS) then (kp is NM)(ki is PM)(kd is PS) (1)

48. If (e is PB) and (ec is PM) then (kp is NB)(ki is PB)(kd is PS) (1)

49 If (e is PB) and (ec is PB) then (kp is NB)(ki is PB)(kd is PB) (1)

3.4 模糊免疫 PID 控制算法

3.4.1 模糊免疫 PID 控制算法原理

常规增量式 PID 控制器离散形式如下：

$$\begin{aligned} u(k) &= u(k-1)+k_p(e(k)-e(k-1))+k_ie(k)+k_d(e(k)-2e(k-1)+e(k-2)) \\ &= u(k-1)+k_p((e(k)-e(k-1))+\frac{k_i}{k_p}e(k)+\frac{k_d}{k_p}(e(k)-2e(k-1)+e(k-2)) \end{aligned} \tag{3.12}$$

式中，k_p,k_i,k_d 分别为比例、积分和微分系数。

P 控制器的控制算法为：

$$u(k)=k_pe(k) \tag{3.13}$$

免疫 PID 控制器是借鉴生物系统的免疫机理而设计出的一种非线性控制器。免疫是生物体的一种特性生理反应。生物的免疫系统对于外来侵犯的抗原，可产生相应的抗体来抵御。抗原和抗体结合后，会产生一系列的反应，通过吞噬作用或产生特殊酶的作用而毁坏抗原。生物的免疫系统由淋巴细胞和抗体分子组成，淋巴细胞又由胸腺产生的 T 细胞（分别为辅助细胞 T_H 和抑制细胞 T_S）和骨髓产生的 B 细胞组成。当抗原侵入机体并经周围细胞消化后，将信息传递给 T 细胞，即传递给 T_H 细胞和 T_S 细胞，然后刺激 B 细胞。B 细胞产生抗体以消除抗原。当抗原较多时，机体内的 T_H 细胞也较多，而 T_S 细胞却较少，从而会产生较多的 B 细胞。随着抗原的减少，体内 T_S 细胞增多，它抑制了 T_H 细胞的产生，则 B 细胞也随着减少。经过一段时间间隔后，免疫反馈系统便趋于平衡。抑制机理和主反馈机理之间的相互协作，是通过免疫反馈机理对抗原的快速反应和稳定免疫系统完成的。

免疫系统虽然十分复杂，但其抵御抗原的自适应能力却是十分明显的。生物信息系统的这些智能行为，为科学和工程领域提供了各种理论参考和技术方法。基于上述免疫反馈原理，提出了免疫 PID 控制器：假设第 k 代的抗原数量为 $\varepsilon(k)$，由抗原刺激的 T_H 细胞的输出为 $T_H(k)$，T_S 细胞对 B 细胞的影响为 $T_S(k)$，则 B 细胞接收的总刺激为：

$$S(k)=T_H(k)-T_S(k) \tag{3.14}$$

式中，$T_H(k)=k_1\varepsilon(k), T_S(k)=k_2f(S(k),\Delta S(k))\varepsilon(k)$。

若以抗原的数量 $\varepsilon(k)$ 作为偏差 $e(k)$，B 细胞接收的总刺激 $S(k)$ 作为控制输入 $u(k)$，则 $\Delta S(k)=\Delta u(k)$。有如下的反馈控制规律：

$$u(k)=K(1-\eta f(u(k),\Delta u(k)))e(k)=k_{p1}e(k) \tag{3.15}$$

式中，$k_{p1}=K(1-\eta f(u(k),\Delta u(k)))$，$K=k_1$ 为控制反应速度，$\eta=\frac{k_2}{k_1}$ 为控制稳定效果，$f(\cdot)$ 为一个选定的非线性函数，$f(\cdot)$ 表示细胞抑制刺激能力的大小。

利用模糊规则可逼近非线性函数 $f()$：每个输入变量被两个模糊集模糊化，分别是“正”（P）和“负”（N）；输出变量被三个模糊集模糊化，分别是“正”（P）、零“Z”和负（N）。

以上隶属度函数都定义在整个$(-\infty,+\infty)$区间。按“细胞接受的刺激越大，则抑制能力越小”及“细胞接受的刺激越小，则抑制能力越大”的原则，可采用以下四条模糊规则：

（1）If u is P and Δu is P then f(u, Δu) is N（1）

（2）If u is P and Δu is N then f(u, Δu) is Z（1）

（3）If u is N and Δu is P then f(u, Δu) is Z（1）

（4）If u is N and Δu is N then f(u, Δu) is P（1）

在各规则中，使用Zadeh的模糊逻辑 AND 操作，并采用“centroid”反模糊化方法得到模糊控制器的输出$f()$。

基于免疫反馈原理的控制器实际上就是一个非线性 P 控制器，其比例系数$k_{p1}=K(1-\eta f(u(k),\Delta u(k)))$随控制器输出的变化而变化，其中 K 为增益，则免疫 PID 控制器的输出为：

$$
\begin{aligned}
u(k)&=u(k-1)+k_{p1}((e(k)-e(k-1))+\frac{k_i}{k_p}e(k)+\frac{k_d}{k_p}(e(k)-2e(k-1)+e(k-2))) \\
&=u(k-1)+k_{p1}((e(k)-e(k-1))+k_i'e(k)+k_d'(e(k)-2e(k-1)+e(k-2)))
\end{aligned}
\tag{3.16}
$$

3.4.2 仿真程序及分析

仿真实例

设被控对象为一个延迟系统：

$$G_p(s)=\frac{1}{60s+1}e^{-80s}$$

采样时间为 20s，采用免疫 PID 控制器式（3.16），取 $K=0.30$，$\eta=0.80$，$k_i'=0.30, k_d'=0.30$。输入的指令信号为$1.0\text{sgn}(\sin(3\pi t))$，仿真时间为 1000 个采样点。为了测试控制器的鲁棒性，在第 500 个采样时间时加入一个干扰，免疫 PID 控制方波跟踪结果如图 3-34 和图 3-35 所示。仿真结果表明，免疫 PID 控制具有很好的控制效果和较高的鲁棒性。非线性函数$f()$为模糊控制的输出，模糊控制输入、输出隶属函数如图 3-36～图 3-38 所示。

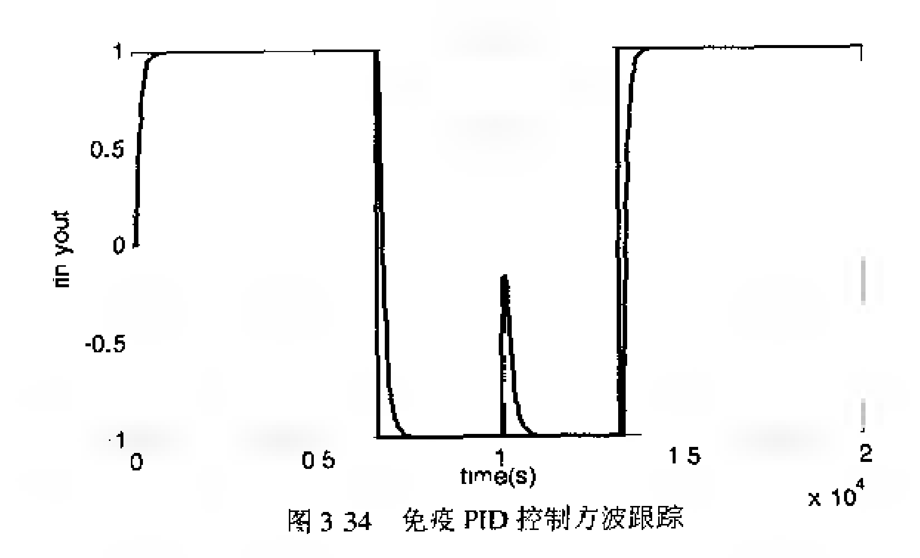


图 3-34 免疫 PID 控制方波跟踪

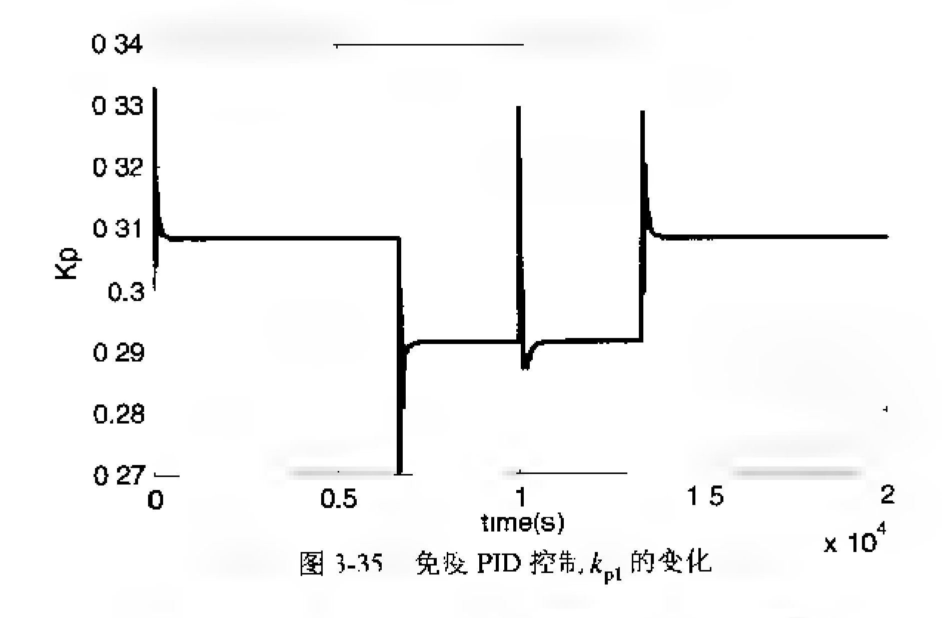


图 3-35　免疫 PID 控制 k_{p1} 的变化

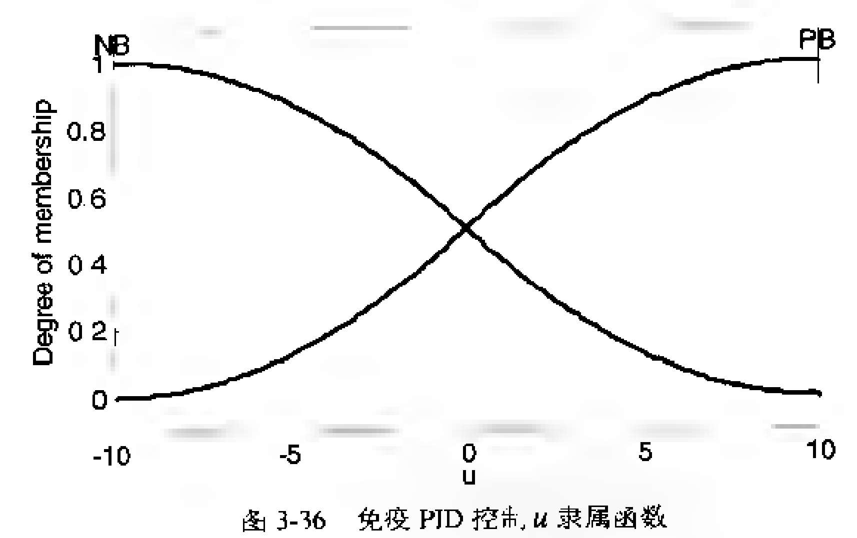


图 3-36　免疫 PID 控制 u 隶属函数

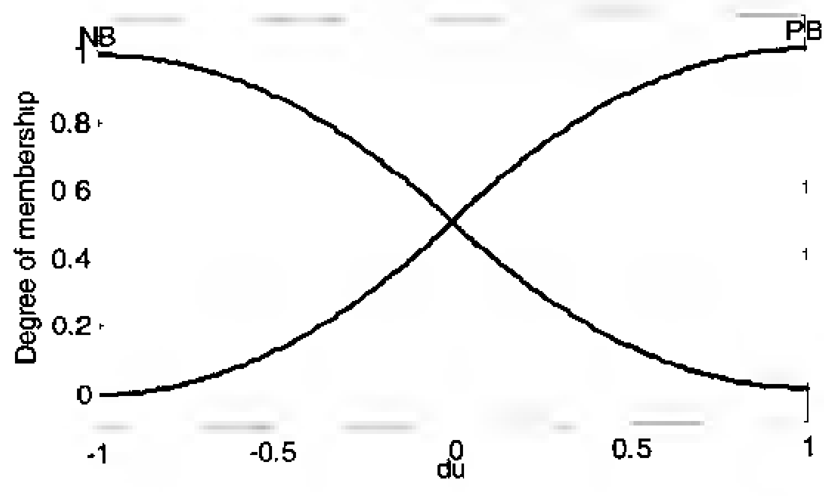


图 3-37　免疫 PID 控制 du 隶属函数

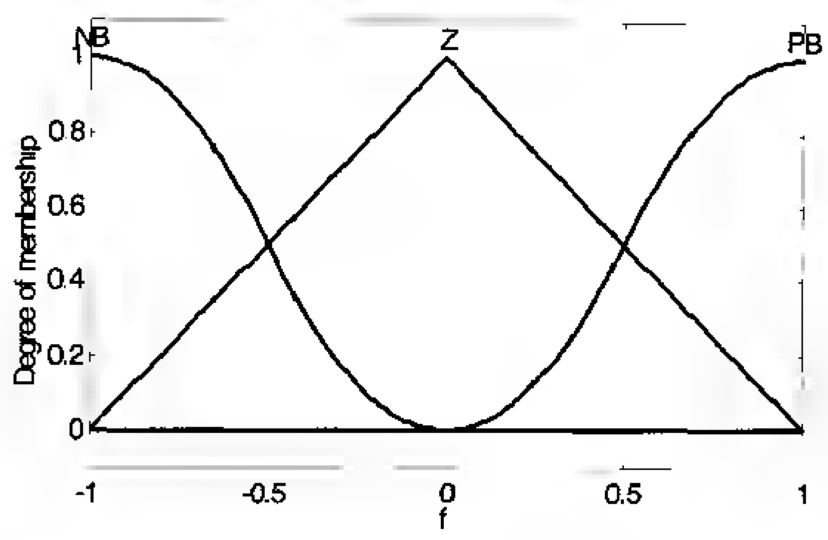


图 3-38　免疫 PID 控制 $f()$ 隶属函数

仿真程序：chap3_7.m。

```
%Fuzzy Immune PID Control
clear all;
close all;

a=newfis('fuzz_ljk');

f1=1.0;
a=addvar(a,'input','u',[-f1*1,f1*1]);             %Parameter e
a=addmf(a,'input',1,'NB','zmf',[-f1*1,f1*1]);
a=addmf(a,'input',1,'PB','smf',[-f1*1,f1*1]);

f2=1.0;
a=addvar(a,'input','du',[-f2*1,f2*1]);            %Parameter ec
a=addmf(a,'input',2,'NB','zmf',[-f2*1,f2*1]);
a=addmf(a,'input',2,'PB','smf',[-f2*1,f2*1]);

f3=1.0;
a=addvar(a,'output','f',[-f3*1,f3*1]);            %Parameter u
a=addmf(a,'output',1,'NB','zmf',[-f3*1,0]);
a=addmf(a,'output',1,'Z','trimf',[-f3*1,0,f3*1]);
a=addmf(a,'output',1,'PB','smf',[0,f3*1]);

rulelist=[2 2 1 1 1;      % Edit rule base
          2 1 2 1 1;
          1 2 2 1 1;
```

```
        1 1 5 1 1;];

a=addrule(a,rulelist);
%showrule(a)             % Show fuzzy rule base

a1=setfis(a,'DefuzzMethod','centroid');  % Defuzzy
writefis(a1,'ljk');      % save to fuzzy file "ljk.fis" which can be
                         % simulated with fuzzy tool
a2=readfis('ljk');
%plotfis(a2);

%%%%%%%%%%%%%%%%%% Using Fuzzy Controller %%%%%%%%%%%%%%%%%%
ts=20;
sys=tf([1],[60,1],'inputdelay',80);
dsys=c2d(sys,ts,'zoh');
[num,den]=tfdata(dsys,'v');

u_1=0;u_2=0;u_3=0;u_4=0;u_5=0;
y_1=0;
e_1=0;e_2=0;
for k=1:1:1000
time(k)=k*ts;

rin(k)=1.0*sign(sin(3*pi*k*0.001));

%Linear model
yout(k)=-den(2)*y_1+num(2)*u_5;

e(k)=rin(k)-yout(k);
ec(k)=e(k)-e_1;

f(k)=evalfis([u_1 u_1-u_2],a2);

K=0.30;
xite=0.80;
Kp(k)=K*(1-xite*f(k));

u(k)=u_1+Kp(k)*((e(k)-e_1)+0.3*e(k)+0.3*(e(k)-2*e_1+e_2));
```

```
if k==500
   u(k)=u(k)+1.0;
end

%Return of parameters
u_5=u_4;u_4=u_3;u_3=u_2;u_2=u_1;u_1=u(k);
y_1=yout(k);
e_2=e_1;
e_1=e(k);
end
figure(1);
plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,e,'r');
xlabel('time(s)');ylabel('error');
figure(3);
plot(time,u,'r');
xlabel('time(s)');ylabel('u');
figure(4);
plot(time,Kp,'r');
xlabel('time(s)');ylabel('Kp');
figure(5);
plotmf(a,'input',1);
figure(6);
plotmf(a,'input',2);
figure(7);
plotmf(a,'output',1);
```

3.5 基于 Sugeno 的模糊控制

3.5.1 Sugeno 模糊模型

传统的模糊系统为 Mamdani 模糊模型，输出为模糊量。

Sugeno 模糊模型的输出隶属函数为 constant 或 linear，其函数形式为：

$$
\begin{aligned}
y &= a \\
y &= ax + b
\end{aligned}
\tag{3.17}
$$

它与 Mamdani 模型的区别在于：

（1）输出变量为常量或线性函数；

（2）输出为精确量。

Sugeno 型的模糊推理系统非常适合于分段线性控制系统，例如在导弹、飞行器的控制中，可根据高度和速度（马赫数）建立 Sugeno 型的模糊推理系统，实现性能良好的线性控制。

3.5.2 Sugeno 模糊模型的建立

设输入 $X \in [0,5]$，$Y \in [0,10]$，将它们模糊化为两个模糊量：小，大。输出 Z 为输入 (x,y) 的线性函数，模糊规则为：

If X为small and Y为small then $Z=-x+y-3$

If X为small and Y为big then $Z=x+y+1$

If X为big and Y为small then $Z=-2y+2$

If X为big and Y为big then $Z=2x+y-6$

仿真程序见 chap3_8.m。模糊推理系统的输入隶属函数曲线及输入/输出曲线如图 3-39 和图 3-40 所示。

通过命令 showrule(ts2)可显示模糊控制规则，共以下 4 条：

If (X is small) and (Y is small) then (Z is first area) (1)

If (X is small) and (Y is big) then (Z is second area) (1)

If (X is big) and (Y is small) then (Z is third area) (1)

If (X is big) and (Y is big) then (Z is fourth area) (1)

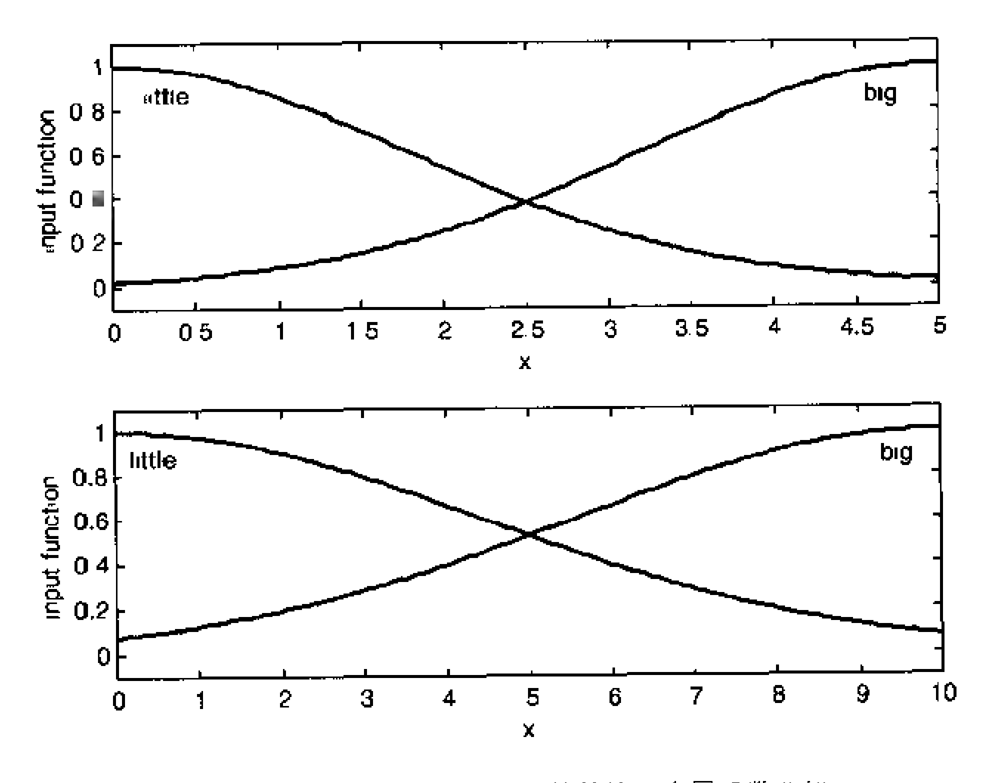


图 3-39　Sugeno 模糊推理系统的输入隶属函数曲线



图 3-40 Sugeno 模糊推理系统的输入/输出曲线

仿真程序：chap3_8.m。

```
%Sugeno type fuzzy model
clear all;
close all;

ts2=newfis('ts2','sugeno');

ts2=addvar(ts2,'input','X',[0 5]);
ts2=addmf(ts2,'input',1,'little','gaussmf',[1.8 0]);
ts2=addmf(ts2,'input',1,'big','gaussmf',[1.8 5]);

ts2=addvar(ts2,'input','Y',[0 10]);
ts2=addmf(ts2,'input',2,'little','gaussmf',[4.4 0]);
ts2=addmf(ts2,'input',2,'big','gaussmf',[4.4 10]);

ts2=addvar(ts2,'output','Z',[-3 15]);
ts2=addmf(ts2,'output',1,'first area','linear',[-1 1 -3]);
ts2=addmf(ts2,'output',1,'second area','linear',[1 1 1]);
ts2=addmf(ts2,'output',1,'third area','linear',[-2 0 3]);
ts2=addmf(ts2,'output',1,'fourth area','linear',[2 1 -6]);

rulelist=[1 1 1 1 1;
          1 2 2 1 1;
          2 1 3 1 1;
          2 2 4 1 1];
```

```
ts2=addrule(ts2,rulelist);
showrule(ts2);

disp('-------------------------------------------------------------');
disp('     fuzzy controller table:e=[-5,+5],ec=[-10,+10]     ');
disp('-------------------------------------------------------------');

Ulist=zeros(11,21);

for i=1:1:6
   for j=1:1:11
      e(i)=i-1;
      ec(j)=j-1;
      Ulist(i,j)=evalfis([e(i),ec(j)],ts2);
   end
end
Ulist=ceil(Ulist)

figure(1);
subplot 211;
plotmf(ts2,'input',1);
xlabel('x'),ylabel('input function');
subplot 212;
plotmf(ts2,'input',2);
xlabel('x'),ylabel('input function');

figure(2);
gensurf(ts2);
xlabel('x'),ylabel('y'),zlabel('z');
```

3.5.3 基于 Sugeno 的倒立摆模糊控制

倒立摆的动力学方程为：

$$\begin{aligned}\dot{x}_1 &= x_2\\ \dot{x}_2 &= \frac{g\sin(x_1)-amlx_2^2\sin(2x_1)/2-a\cos(x_1)u}{4/3l-aml\cos^2 x_1}\end{aligned} \tag{3.18}$$

式中，x_1 表示摆与垂直线的夹角，$x_1=\theta$，x_2 表示摆的旋转角速度，$x_2=\dot{\theta}$，$g=9.8\text{m/s}^2$ 为重力加速度，m 为倒立摆的质量，$2l$ 为摆长，$a=l/(m+M)$，M 为小车质量。

如果摆角很小，则倒立摆的动力学方程可简化为：

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = \frac{gx_1 - amlx_2^2 x_1 - au}{4/3l - aml} \quad (3\text{-}19)$$

取倒立摆参数 $m = 2\text{kg}$，$M = 8\text{kg}$，$l = 0.5\text{m}$，在 (x_1, x_2) 平面上对倒立摆动力学方程进行模糊分割，实现倒立摆模型的局部线性化。

倒立摆角度范围为 $[-15, +15]$ 度，即 $[-15, +15] \times \frac{\pi}{180}$ 弧度，角速度范围为 $[-200, +200]$ 度/秒，即 $[-200, +200] \times \frac{\pi}{180}$ 弧度/秒。它们的隶属函数如图 3-41 和图 3-42 所示。

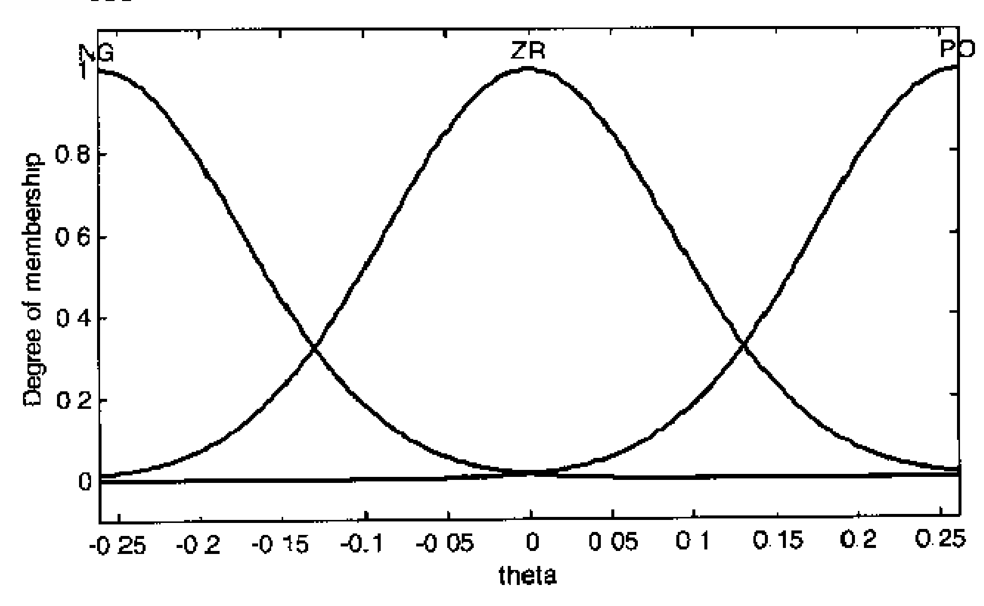


图 3-41 角度 θ（弧度）的隶属度曲线

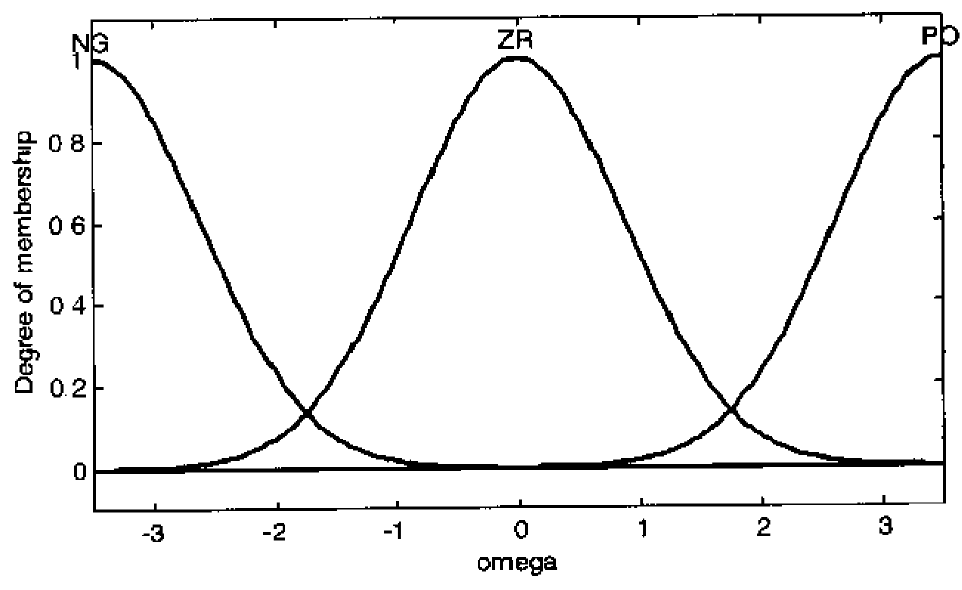


图 3-42 角速度 $\dot{\theta}$（弧度/秒）的隶属度曲线

令 $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$，可得到 5 个线性化方程，表示为以下 7 条 Sugeno 型模糊规则：

R1：If x_1 为 ZR and x_2 为 ZR then $\dot{x} = A_1 x + B_1 u$

R2：If x_1 为 ZR and x_2 为 NG 或 PO then $\dot{x} = A_2 x + B_2 u$

R3. If x 为 NG 或 PO and x_2为 ZR then $\dot{x} = A_3x + B_3u$

R4. If x 为 PO and x_2为 PO then $\dot{x} = A_4x + B_4u$

R5: If x_1为 NG and x_2为 NG then $\dot{x} = A_4x + B_4u$

R6: If x_1为 PO and x_2为 NG then $\dot{x} = A_5x + B_5u$

R7: If x_1为 NG and x_2为 PO then $\dot{x} = A_5x + B_5u$

对 R1，当x_1和x_2都为 ZR 时，倒立摆的动力学方程为：

$$\dot{x}_2 = \frac{gx_1 - au}{4/3l - aml} \tag{3.20}$$

对 R2，当x_1为 ZR，x_2为 NG 或 PO 时，$x_2 = \pm 200 \times \frac{\pi}{180}$，倒立摆的动力学方程为：

$$\dot{x}_2 = \frac{gx_1 - amlx_2^2x_1 - au}{4/3l - aml} \tag{3.21}$$

对 R3，当x_1为 NG 或 PO，x_2为 ZR 时，$x_1 = \pm 15 \times \frac{\pi}{180}$，倒立摆的动力学方程为：

$$\dot{x}_2 = \frac{gx_1 - a\cos(x_1)u}{4/3l - aml\cos^2(x_1)} \tag{3.22}$$

对 R4，当x_1为 PO，x_2为 PO 时，$x_1 = 15 \times \frac{\pi}{180}$，$x_2 = 200 \times \frac{\pi}{180}$，倒立摆的动力学方程为：

$$\dot{x}_2 = \frac{gx_1 - amlx_2\sin(2x_1)/2 \times x_2 - a\cos(x_1)u}{4/3l - aml\cos^2(x_1)} \tag{3.23}$$

对 R5，当x_1为 NG，x_2为 NG 时，$x_1 = -15 \times \frac{\pi}{180}$，$x_2 = -200 \times \frac{\pi}{180}$，倒立摆的动力学方程同式（3.23）。

对 R6，当x_1为 PO，x_2为 NG 时，$x_1 = 15 \times \frac{\pi}{180}$，$x_2 = -200 \times \frac{\pi}{180}$，倒立摆的动力学方程为：

$$\dot{x}_2 = \frac{gx_1 - amlx_2\sin(2x_1)/2 \times x_2 - a\cos(x_1)u}{4/3l - aml\cos^2(x_1)} \tag{3.24}$$

对 R7，当x_1为 NG，x_2为 PO 时，$x_1 = -15 \times \frac{\pi}{180}$，$x_2 = 200 \times \frac{\pi}{180}$，倒立摆的动力学方程同式（3.24）。

将倒立摆实际参数值及x_1、x_2分别带入式（3.20）～式（3.24），得：

$$A_1 = \begin{bmatrix} 0 & 1 \\ 15.8919 & 0 \end{bmatrix},\quad A_2 = \begin{bmatrix} 0 & 1 \\ 14.9039 & 0 \end{bmatrix},\quad A_3 = \begin{bmatrix} 0 & 1 \\ 15.806 & 0 \end{bmatrix}$$

$$A_4 = \begin{bmatrix} 0 & 1 \\ 15.806 & -0.0704 \end{bmatrix},\quad A_5 = \begin{bmatrix} 0 & 1 \\ 15.806 & 0.0704 \end{bmatrix}$$

$$B_1 = \begin{bmatrix} 0 \\ -0.0811 \end{bmatrix},\quad B_2 = \begin{bmatrix} 0 \\ 0.0811 \end{bmatrix},\quad B_3 = \begin{bmatrix} 0 \\ -0.0779 \end{bmatrix}$$

$$B_4 = \begin{bmatrix} 0 \\ -0.0779 \end{bmatrix}, \quad B_5 = \begin{bmatrix} 0 \\ -0.0779 \end{bmatrix}$$

选择期望的闭环极点($-10\pm10i$)，采用 $u=-Fx$ 的反馈控制，利用极点配置函数 place(A,B,P)，可分别得到 5 个线性化方程的反馈增益矩阵 F：

$F_1 = [2662.7 \quad 246.7]$，$F_2 = [2650.5 \quad 246.7]$，$F_3 = [2770.5 \quad 256.8]$

$F_4 = [-2770.5 \quad -255.9]$，$F_5 = [-2770.5 \quad 257.7]$

根据倒立摆的模糊建模过程，可以设计 PD 型模糊控制器，其模糊规则为：

If x_1为 ZR and x_2为 ZR then $u = F_1x$

If x_1为 ZR and x_2为 NG 或 PO then $u = F_2x$

If x_1为 NG 或 PO and x_2为 ZR then $u = F_3x$

If x_1为 PO and x_2为 PO then $u = F_4x$

If x_1为 NG and x_2为 NG then $u = F_4x$

If x_1为 PO and x_2为 NG then $u = F_5x$

If x_1为 NG and x_2为 PO then $u = F_5x$

通过命令 showrule(tc) 可显示模糊控制器“tc.fis”的模糊规则，共 8 条：

```
1. If (theta is NG) and (omega is NG) then (u is No.4) (1)
2. If (theta is NG) and (omega is ZR) then (u is No.3) (1)
3. If (theta is NG) and (omega is PO) then (u is No.5) (1)
4. If (theta is ZR) and (omega is NG) then (u is No.2) (1)
5. If (theta is ZR) and (omega is ZR) then (u is No.1) (1)
6. If (theta is PO) and (omega is NG) then (u is No.5) (1)
7. If (theta is PO) and (omega is ZR) then (u is No.3) (1)
8. If (theta is PO) and (omega is PO) then (u is No.4) (1)
```

通过命令 showrule(model) 可显示模糊控制器“tc.fis”的模糊规则，共 9 条：

```
1. If (theta is NG) and (omega is NG) then (d_theta is No.4)(d_omega is No.4) (1)
2. If (theta is NG) and (omega is ZR) then (d_theta is No.3)(d_omega is No.3) (1)
3. If (theta is NG) and (omega is PO) then (d_theta is No.5)(d_omega is No.5) (1)
4. If (theta is ZR) and (omega is NG) then (d_theta is No.2)(d_omega is No.2) (1)
5. If (theta is ZR) and (omega is ZR) then (d_theta is No.1)(d_omega is No.1) (1)
6. If (theta is ZR) and (omega is PO) then (d_theta is No.2)(d_omega is No.2) (1)
7. If (theta is PO) and (omega is NG) then (d_theta is No.5)(d_omega is No.5) (1)
8. If (theta is PO) and (omega is ZR) then (d_theta is No.3)(d_omega is No.3) (1)
9. If (theta is PO) and (omega is PO) then (d_theta is No.4)(d_omega is No.4) (1)
```

倒立摆的初始状态为（0.20，0）。仿真程序由倒立摆线性化程序 chap3_9f.m 和控制程序 chap3_9.m 两部分组成。在控制程序中，分别实现了用于倒立摆建模的 TS 模糊系统“model.fis”及用于控制的 TS 型模糊控制器“tc.fis”。倒立摆的摆角、角速度及控制器输出的仿真结果如图 3-43 和图 3-44 所示。

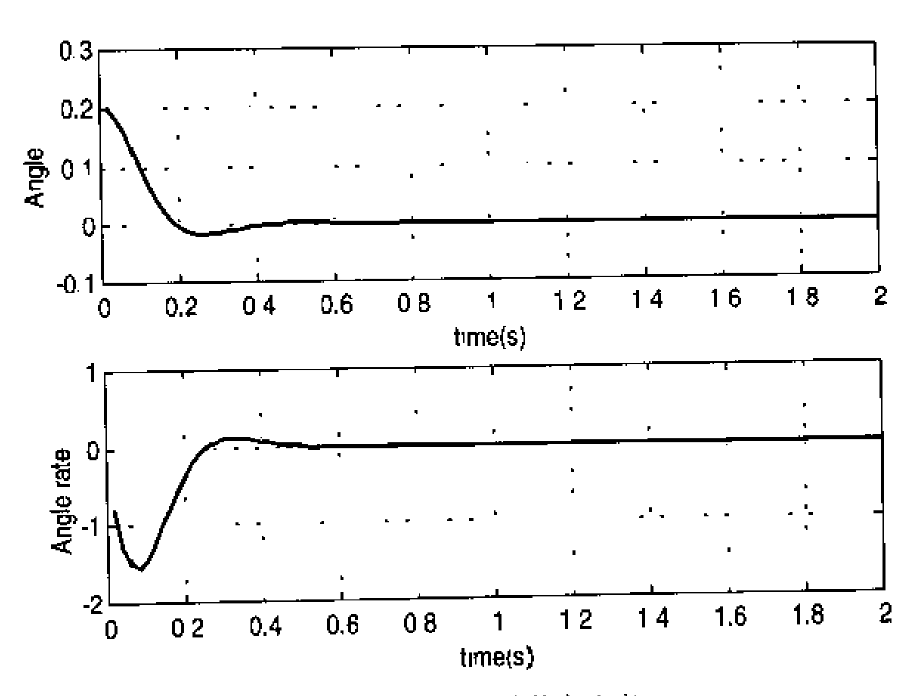


图 3-43 摆角的状态响应

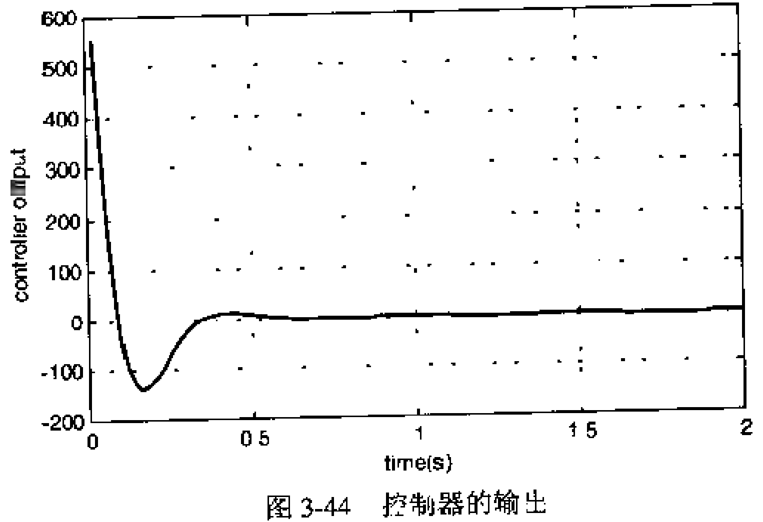


图 3-44 控制器的输出

线性化程序：**chap3_9f.m**。

```
%Local linearization for single inverted pendulum
clear all;
close all;

g 9.8;
m 2;
M=8;
l 0.5;
a=1/(m+M);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Equation 1:
a1 g 4/3*l-a*m*l ,
A1 [0 1;
   a1 0]

b1 --a (4 3*l a*m*l ;
B1-[0;b1,
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Equation 2:
x2 200*pi 180;
a2 g a*m*l*x2^2, (4/3*l-a*m*l ;
A2-[0 1;
   a2 0]

b2-b1;
B2 [0;b2]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Equation 3:
%x1 -15*pi 180;
x1-15*pi/180;

a3 g (4 3*l-a*m*l*(cos(x1  ^2
A3 [0 1;
   a3 0]
b3- a*cos(x1) (4 3*l a*m*l*(cos x1,)^2
B3 [0;b3]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Equation 4:
x1-15*pi/180;
x2-200*pi/180;

a41 g (4/3*l-a*m*l* cos(x1,)^2
a42--a*m*l*x2*sin(2*x1)*0.5 (4 3*l a*m*l*(cos(x1  ^2)
A4 [0  1,
   a41 a42]
b4 b3;
B4 [0,b4]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Equation 5:
x1 -15*pi 180;
x2 200*pi 180;
```

```
a51=g/(4/3*l-a*m*l*cos(x1)^2);
a52=-a*m*l*x2*sin(2*x1)*0.5/(4/3*l-a*m*l*(cos(x1))^2);
A5=[0 1;
   a51 a52];
b5=b3;
B5=[0;b5];
```

控制主程序：chap3_9.m

```
%Sugeno type fuzzy control for single inverted pendulum
close all;

P=[-10-10i; -10+10i];    %Stable pole point

F1=place(A1,B1,P);
F2=place(A2,B2,P);
F3=place(A3,B3,P);
F4=place(A4,B4,P);
F5=place(A5,B5,P);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tc=newfis('tc','sugeno');
tc=addvar(tc,'input','theta',[-15,15]*pi/180);
tc=addmf(tc,'input',1,'NG','gaussmf',[5,-15]*pi/180);
tc=addmf(tc,'input',1,'ZR','gaussmf',[5,0]*pi/180);
tc=addmf(tc,'input',1,'PO','gaussmf',[5,15]*pi/180);

tc=addvar(tc,'input','omega',[-200,200]*pi/180);
tc=addmf(tc,'input',2,'NG','gaussmf',[50,-200]*pi/180);
tc=addmf(tc,'input',2,'ZR','gaussmf',[50,0]*pi/180);
tc=addmf(tc,'input',2,'PO','gaussmf',[50,200]*pi/180);

tc=addvar(tc,'output','u',[-300,0]);
tc=addmf(tc,'output',1,'No.1','linear',[F1(1),F1(2)]);
tc=addmf(tc,'output',1,'No.2','linear',[F2(1),F2(2)]);
tc=addmf(tc,'output',1,'No.3','linear',[F3(1),F3(2)]);
tc=addmf(tc,'output',1,'No.4','linear',[F4(1),F4(2)]);
tc=addmf(tc,'output',1,'No.5','linear',[F5(1),F5(2)]);

rulelist=[1 1 4 1 1;
          1 2 5 1 1;
```

```
         1 3 5 1 1;
         2 1 2 1 1;
         2 2 1 1 1;
         3 1 5 1 1;
         3 2 4 1 1;
         3 3 4 1 1];
tc=addrule(tc,rulelist);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
model=newfis('model','sugeno');
model=addvar(model,'input','theta',[-15,15]*pi/180);
model=addmf(model,'input',1,'NG','gaussmf',[5,-15]*pi/180);
model=addmf(model,'input',1,'ZR','gaussmf',[5,0]*pi/180);
model=addmf(model,'input',1,'PO','gaussmf',[5,15]*pi/180);

model=addvar(model,'input','omega',[-200,200]*pi/180);
model=addmf(model,'input',2,'NG','gaussmf',[50,-200]*pi/180);
model=addmf(model,'input',2,'ZR','gaussmf',[50,0]*pi/180);
model=addmf(model,'input',2,'PO','gaussmf',[50,200]*pi/180);

model=addvar(model,'input','u',[-5,5]);
model=addmf(model,'input',3,'Any','gaussmf',[1.5,5]);

model=addvar(model,'output','d_theta',[-200,200]*pi/180);
model=addmf(model,'output',1,'No.1','linear',[0 1 0 0]);
model=addmf(model,'output',1,'No.2','linear',[0 1 0 0]);
model=addmf(model,'output',1,'No.3','linear',[0 1 0 0]);
model=addmf(model,'output',1,'No.4','linear',[0 1 0 0]);
model=addmf(model,'output',1,'No.5','linear',[0 1 0 0]);

model=addvar(model,'output','d_omega',[-200,200]*pi/180);
model=addmf(model,'output',2,'No.1','linear',[A1(2,1),0,B1(2),0]);
model=addmf(model,'output',2,'No.2','linear',[A2(2,1),0,B2(2),0]);
model=addmf(model,'output',2,'No.3','linear',[A3(2,1),0,B3(2),0]);
model=addmf(model,'output',2,'No.4','linear',[A4(2,1),A4(2,2),B4(2),0]);
model=addmf(model,'output',2,'No.5','linear',[A5(2,1),A5(2,2),B5(2),0]);

rulelist1=[1 1 0 4 4 1 1;
           1 2 0 3 3 1 1;
           1 3 0 5 5 1 1;
           2 1 0 2 2 1 1;
```

```
            2 2 0 1 1 1 1;
            2 3 0 2 2 1 1;
            3 1 0 5 5 1 1;
            3 2 0 3 3 1 1;
            3 3 0 4 4 1 1];
model=addrule(model,rulelist1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ts=0.020;
x=[0.20,0];  %Initial state

for k=1:1:100
   time(k)=k*ts;
   u(k,:)=(-1)*evalfis([x(1),x(2)],tc);   %Using feedback control

   k0=evalfis([x(1),x(2),u(k,:)],model)'; %Using fuzzy T-S model
   x=x+ts*k0;

   y1(k)=x(1);
   y2(k)=x(2);
end
figure(1);
subplot(211);
plot(time,y1),grid on;
xlabel('time(s)'),ylabel('Angle');
subplot(212);
plot(time,y2),grid on;
xlabel('time(s)'),ylabel('Angle rate');

figure(2);
plot(time,u),grid on;
xlabel('time(s)'),ylabel('controller output');

figure(3);
plotmf(tc,'input',1);
figure(4);
plotmf(tc,'input',2);

showrule(tc),
showrule(model);
```

3.6 基于控制规则表的模糊 PD 控制

3.6.1 模糊控制器的原理

模糊控制规则为：

$$\text{Rule } ij\text{: IF } e=\mu_i \text{ and } \dot{e}=\mu_j \text{ THEN } u=u_{ij} \tag{3.25}$$

采用乘积推理机，规则前部分的隶属函数为：

$$f_{ij}=\mu_i(e)\ \mu_j(\dot{e}) \tag{3.26}$$

式中，$\mu_i(e)$ 和 $\mu_j(\dot{e})$ 分别为 μ_i 和 μ_j 的隶属度。

采用重心方法进行反模糊化，得到模糊控制器：

$$u=\frac{\sum_{i,j} f_{ij}u_{ij}}{\sum_{i,j} f_{ij}} \tag{3.27}$$

式中，u_{ij} 的值由模糊规则表确定。

模糊规则表是根据模糊规则进行设计的，每条规则的输出 u_{ij} 可由模糊推理或根据经验确定。假设 e 和 $\dot{e}$ 各有 3 个隶属函数，共 9 条规则，则模糊规则表的形式如图 3-45 所示。

u_{ij}		e		
		N	Z	P
e	N			
	Z			
	P			

图 3-45 控制规则表

3.6.2 仿真程序及分析

设被控制对象为：

$$G(s)=\frac{133}{s^2+25s}$$

采样时间为 1ms，采用 Z 变换进行离散化，经过 Z 变换后的离散化对象为：

$$\text{yout}(k)=-\text{den}(2)\text{yout}(k-1)-\text{den}(3)\text{yout}(k-2)+\text{num}(2)u(k-1)+\text{num}(3)u(k-2)$$

输入指令信号为 $0.5\sin(\pi t)$。

1. 隶属函数的设计

误差 e 及误差变化率 $\dot{e}$ 分别采用 3 个隶属函数，即：

$$\mu_1(x)=\exp\left[-\left(\frac{x+\pi/6}{\pi/12}\right)^2\right]$$

$$\mu_2(x)=\exp\left[-\left(\frac{x}{\pi/12}\right)^2\right]$$

$$\mu_3(x)=\exp\left[-\left(\frac{x-\pi/6}{\pi/12}\right)^2\right]$$

隶属函数如图 3-46 和图 3-47 所示。

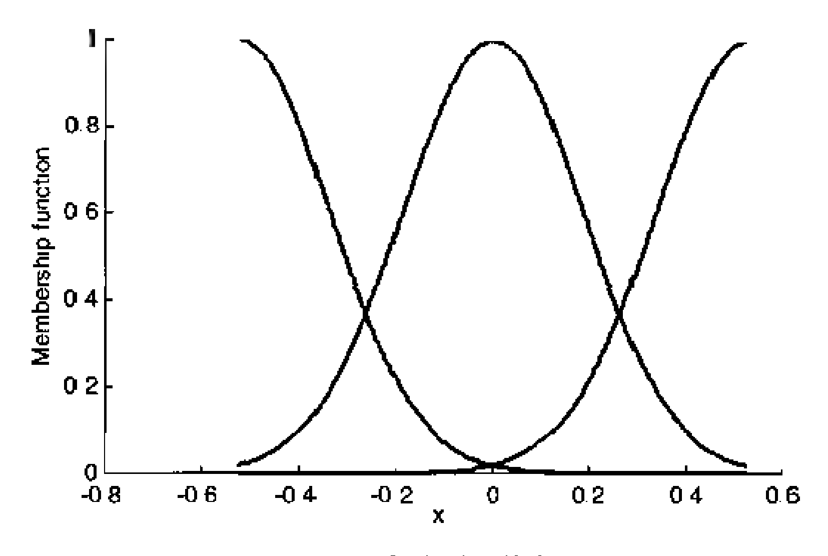


图 3-46 e 的隶属函数度曲线

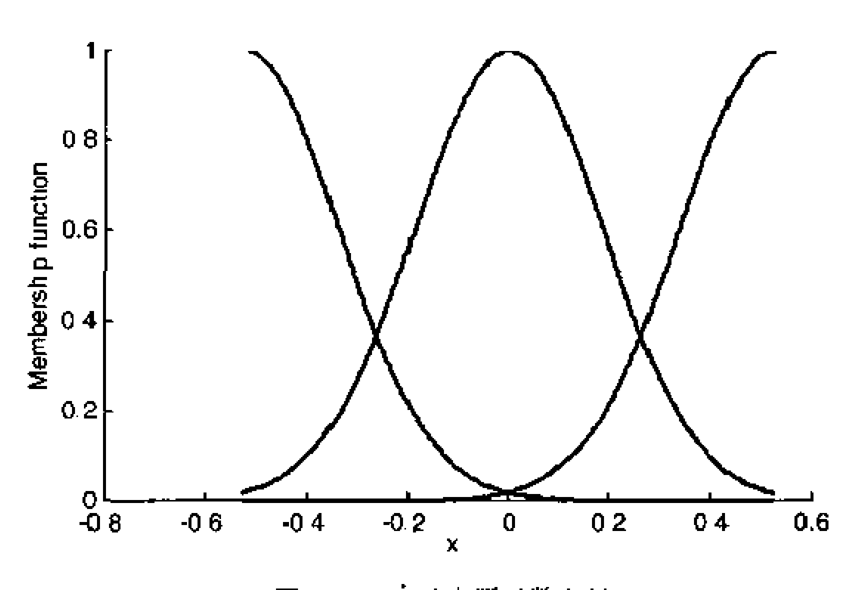


图 3-47 $\dot{e}$ 的隶属函数曲线

隶属函数设计程序：chap3_10plot.m。

```
clear all;
close all;

L1=-pi/6;
L2=pi/6;
```

```
L=L2-L1;

T=L*1/1000;

x=L1:T:L2;
figure(1);
for i=1:1:5
   gs=-[(x+pi/6-(i-1)*pi/6)/(pi/12)].^2;
    u=exp(gs);
    hold on;
    plot(x,u);
end
```

2. 控制规则表的设计

采用经验确定模糊规则表，将控制规则表设计为如图 3-48 所示的形式。

u_o		e N	e Z	e P
e	N	20	10	0
	Z	10	0	10
	P	0	10	20

图 3-48　控制规则表

3. 仿真实验

采用控制律式（3.22），正弦位置跟踪结果如图 3-49 所示。

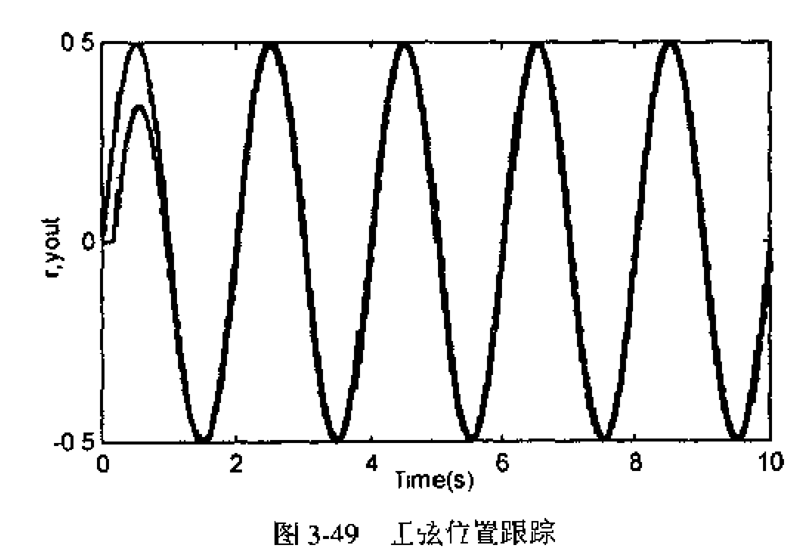


图 3-49　正弦位置跟踪

控制器性能的好坏决定了控制规则表，而采用经验很难确定控制性能高的规则表。可采

用定性分析及遗传算法对规则表中规则的数目和规则表中的数值进行优化。

仿真程序一

仿真程序一：chap3_10.m。

```
%PD Type Fuzzy Controller Design
clear all;
close all;

ts=0.001;

sys=tf(133,[1,25,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

e_1=0;
u_1=0;u_2=0;
y_1=0;y_2=0;

for k=1:1:10000
time(k)=k*ts;

rin(k)=0.5*sin(1*pi*k*ts);

yout(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;

e(k)=rin(k)-yout(k);
de(k)=(e(k)-e_1)/ts;

for l1=1:1:3
   gs1=-[(e(k)+pi/6-(l1-1)*pi/6)/(pi/12)]^2;
    u1(l1)=exp(gs1);
end

for l2=1:1:3
   gs2=-[(de(k)+pi/6-(l2-1)*pi/6)/(pi/12)]^2;
    u2(l2)=exp(gs2);
end

U=[-20 -10 0
   -10 0 10
    0 10 20];

fnum=0;
fden=0;
for i=1:1:3
    for j=1:1:3
```

```
            fnum=fnum+u1(i)*u2(j)*U(i,j);
            fden=fden+u1(i)*u2(j);
        end
    end

    u(k)=fnum/(fden+0.01);

    e_1=e(k);
    u_2=u_1;u_1=u(k);
    y_2=y_1;y_1=yout(k);
    end
    figure(1);
    plot(time,rin,'r',time,yout,'b');
    xlabel('Time(s)');ylabel('r,yout');

    figure(2);
    plot(time,e,'r');
    xlabel('Time(s)');ylabel('e');

    figure(3);
    plot(time,de,'r');
    xlabel('Time(s)');ylabel('de');

    figure(4);
    plot(time,u,'r');
    xlabel('Time(s)');ylabel('u');
```

仿真程序二

为了将程序模块化，把控制器提出来，写成函数的形式，函数名为 chap3_11f，采用控制律式（3.22），正弦位置跟踪仿真结果如图 3-50 所示，仿真结果与仿真程序一相同。

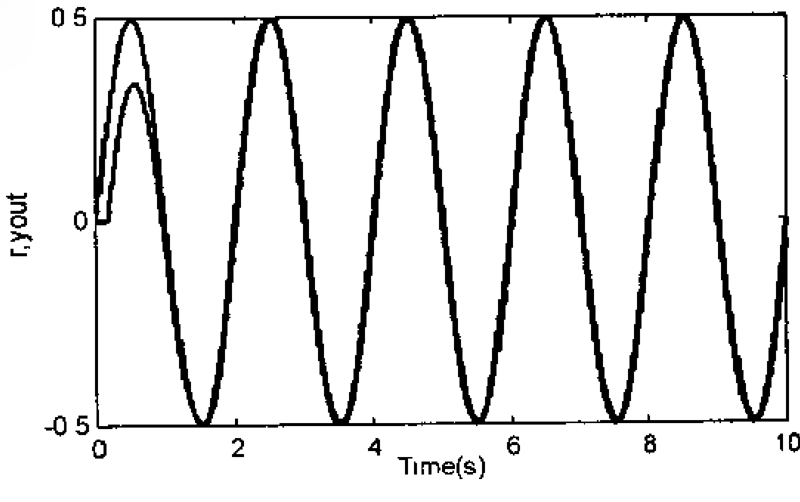


图 3-50 正弦位置跟踪

卜程序：chap3_11 m

```
%PD Type F.zzy Controller Design
clear all·
close all;

ts ;.( 1;

sys tf 133,[1,25,0] ;
dsys c2d s,s ts,'  ;
[n.m den] tfdata dsys, v  ;

e 1 0;
u 1-0; _2 0,
y 1 0;y 2  ,

for k 1:1 100 0
t.me k  k*ts;

rin k   .5*sin 1*pi*k*ts);

yout k   den 2 *y_1 den 3 *y_2+num 2 *u 1+num  ) *  ;

e k) rin(k  yout k);
de k  e(k  e_1  ts;

 k  chap3 11f(e k),de k  ;

e 1=e k);
u_2 u  ;u 1  (k);
y 2=y_1,y 1 yout(k);
end
figure(1 ;
plot(time,rin, r',time,yout,'b' ;
xlabel('Time s)' ;ylabel  r,yout');

figure 2 ;
plot  time,e,'r');
xlabel('Time s)  ,ylabel  e');

figure 3 ;
plot  time,de,'r  ,
```

```
xlabel('Time(s)');ylabel('de');

figure(4);
plot(time,u,'r');
xlabel('Time(s)');ylabel('u');
```

控制器子程序：chap3_11f.m。

```
function [y]=func(x1,x2,x3)

for l1=1:1:3
  gs1=-[(x1+pi/6-(l1-1)*pi/6)/(pi/12)]^2;
    u1(l1)=exp(gs1);
end

for l2=1:1:3
  gs2=-[(x2+pi/6-(l2-1)*pi/6)/(pi/12)]^2;
    u2(l2)=exp(gs2);
end

U=[-20 -10 0
  -10 0 10
   0 10 20];

fnum=0;
fden=0;
for i=1:1:3
    for j=1:1:3
        fnum=fnum+u1(i)*u2(j)*U(i,j);
        fden=fden+u1(i)*u2(j);
    end
end

y=fnum/(fden+0.01);
```

第 4 章　神经 PID 控制

4.1　基于单神经元网络的 PID 智能控制

由具有自学习和自适应能力的单神经元构成的单神经元自适应智能 PID 控制器，不但结构简单，而且能适应环境变化，有较强的鲁棒性。

4.1.1　几种典型的学习规则

（1）无监督 Hebb 学习规则

Hebb 学习是一类相关学习，其基本思想是，如果两个神经元同时被激活，则它们之间的连接强度的增强与它们激励的乘积成正比，以 o_i 表示神经元 i 的激活值，o_j 表示神经元 j 的激活值，w_{ij} 表示神经元 i 和神经元 j 的连接权值，则 Hebb 学习规则可表示为：

$$\Delta w_{ij}(k)=\eta o_j(k)o_i(k) \tag{4.1}$$

式中，η 为学习速率。

（2）有监督的 Delta 学习规则

在 Hebb 学习规则中，引入教师信号，即将 o_j 换成希望输出 d_j 与实际输出 o_j 之差，就构成有监督学习的 Delta 学习规则：

$$\Delta w_{ij}(k)=\eta(d_j(k)-o_j(k))o_i(k) \tag{4.2}$$

（3）有监督的 Hebb 学习规则

将无监督的 Hebb 学习规则和有监督的 Delta 学习规则两者结合起来就构成有监督的 Hebb 学习规则：

$$\Delta w_{ij}(k)=\eta(d_j(k)-o_j(k))o_j(k)o_i(k) \tag{4.3}$$

4.1.2　单神经元自适应 PID 控制

单神经元自适应 PID 控制结构如图 4-1 所示。

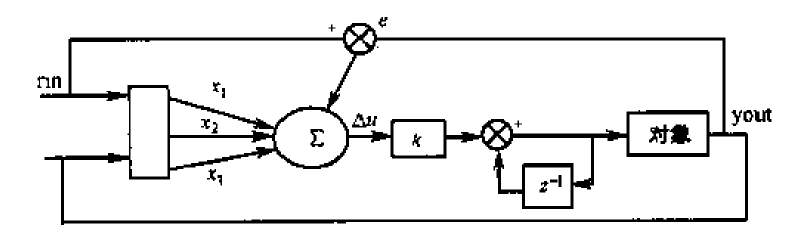


图 4-1　单神经元自适应 PID 控制结构

单神经元自适应控制器是通过对加权系数的调整来实现自适应、自组织功能的，权系数的调整是按有监督的 Hebb 学习规则实现的。控制算法及学习算法为：

$$u(k)=u(k-1)+K\sum_{i=1}^{3}w_i'(k)x_i(k) \tag{4.4}$$

$$w_i'(k)=w_i(k)/\sum_{i=1}^{3}\left|w_i(k)\right| \tag{4.5}$$

$$\begin{aligned} w_1(k)&=w_1(k-1)+\eta_{\mathrm{I}}z(k)u(k)x_1(k)\\ w_2(k)&=w_2(k-1)+\eta_{\mathrm{P}}z(k)u(k)x_2(k)\\ w_3(k)&=w_3(k-1)+\eta_{\mathrm{D}}z(k)u(k)x_3(k) \end{aligned} \tag{4.6}$$

式中，$x_1(k)=e(k)$;

$x_2(k)=e(k)-e(k-1)$;

$x_3(k)=\Delta^2 e(k)=e(k)-2e(k-1)+e(k-2)$;

$z(k)=e(k)$

$\eta_{\mathrm{I}},\eta_{\mathrm{P}},\eta_{\mathrm{D}}$ 分别为积分、比例、微分的学习速率，K 为神经元的比例系数，$K>0$。

对积分 I、比例 P 和微分 D 分别采用了不同的学习速率 $\eta_{\mathrm{I}},\eta_{\mathrm{P}},\eta_{\mathrm{D}}$，以便对不同的权系数分别进行调整。

K 值的选择非常重要。K 越大，则快速性越好，但超调量大，甚至可能使系统不稳定。当被控对象时延增大时，K 值必须减少，以保证系统稳定。K 值选择过小，会使系统的快速性变差。

4.1.3 改进的单神经元自适应 PID 控制

在大量的实际应用中，通过实践表明，PID 参数的在线学习修正主要与 $e(k)$ 和 $\Delta e(k)$ 有关。基于此可将单神经元自适应 PID 控制算法中的加权系数学习修正部分进行修改，即将其中的 $x_i(k)$ 改为 $e(k)+\Delta e(k)$，改进后的算法如下：

$$u(k)=u(k-1)+K\sum_{i=1}^{3}w_i'(k)x_i(k) \tag{4.7}$$

$$w_i'(k)=w_i(k)/\sum_{i=1}^{3}\left|w_i(k)\right|$$

$$\begin{aligned} w_1(k)&=w_1(k-1)+\eta_{\mathrm{I}}z(k)u(k)(e(k)+\Delta e(k))\\ w_2(k)&=w_2(k-1)+\eta_{\mathrm{P}}z(k)u(k)(e(k)+\Delta e(k))\\ w_3(k)&=w_3(k-1)+\eta_{\mathrm{D}}z(k)u(k)(e(k)+\Delta e(k)) \end{aligned}$$

式中，$\Delta e(k)=e(k)-e(k-1)$，$z(k)=e(k)$。

采用上述改进算法后，权系数的在线修正就不完全是根据神经网络学习原理，而是参考实际经验制定的。

4.1.4 仿真程序及分析

仿真实例

设被控制对象为：

$$y(k)=0.368y(k-1)+0.26y(k-2)+0.10u(k-1)+0.632u(k-2)$$

输入指令为一个方波信号：$rin(k)=0.5\text{sgn}(\sin(4\pi t))$，采样时间为1ms，分别采用四种控制律进行单神经元PID控制，即无监督的Hebb学习规则；有监督的Delta学习规则；有监督的Hebb学习规则；改进的Hebb学习规则，跟踪结果如图4-2～图4-5所示。

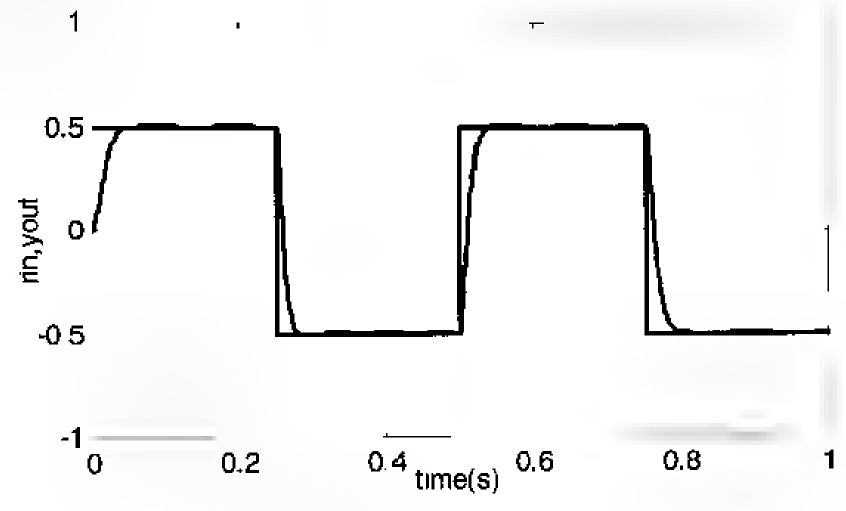


图4-2　基于无监督Hebb学习规则的位置跟踪（M=1）

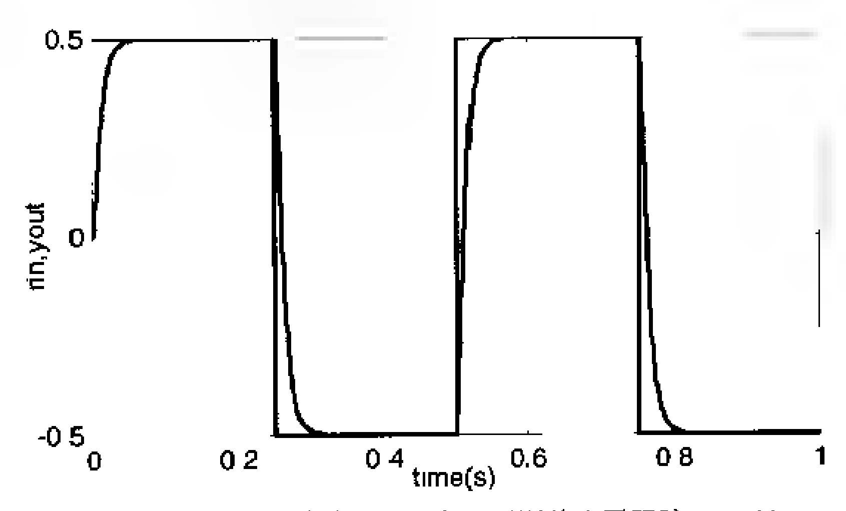


图4-3　基于有监督的Delta学习规则的位置跟踪（M=2）

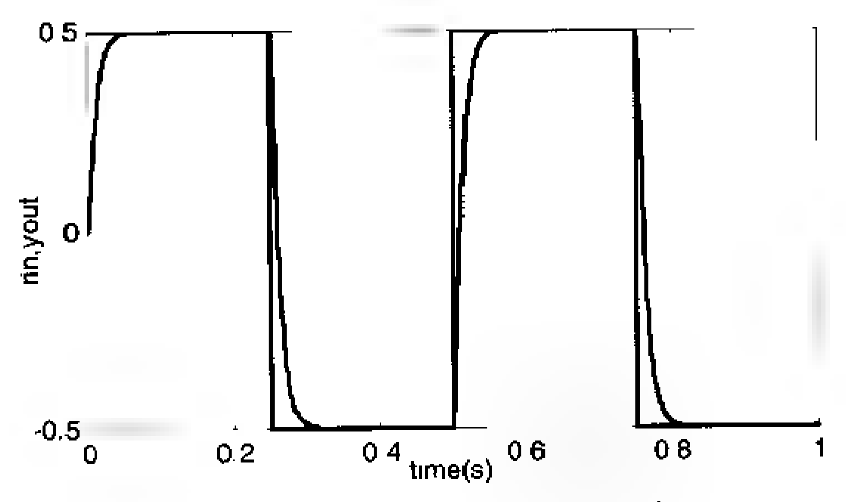


图4-4　基于有监督Hebb学习规则的位置跟踪（M=3）

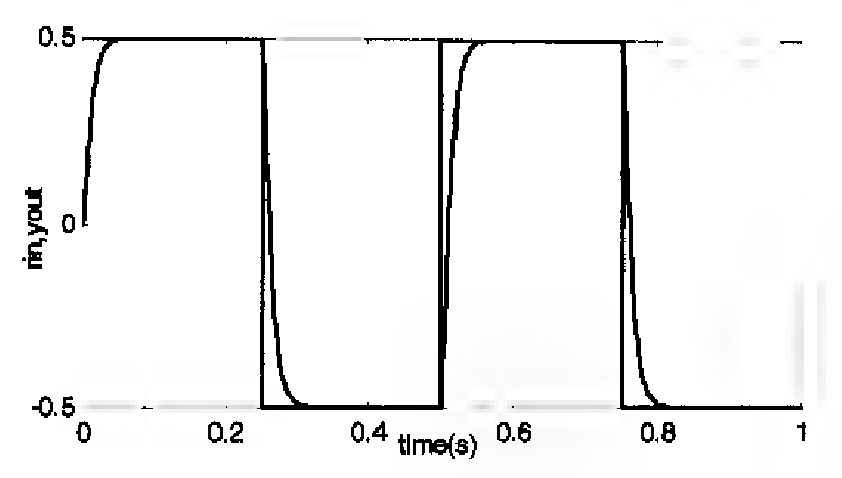


图 4-5　基于改进学习规则的位置跟踪（M=4）

仿真程序：chap4_1.m。

```
%Single Neural Adaptive PID Controller
clear all;
close all;

x=[0,0,0]';

xiteP=0.40;
xiteI=0.35;
xiteD=0.40;

%Initilizing kp,ki and kd
wkp_1=0.10;
wki_1=0.10;
wkd_1=0.10;
%wkp_1=rand;
%wki_1=rand;
%wkd_1=rand;

error_1=0;
error_2=0;
y_1=0;y_2=0;y_3=0;
u_1=0;u_2=0;u_3=0;

ts=0.001;
for k=1:1:1000
    time(k)=k*ts;
    rin(k)=0.5*sign(sin(2*2*pi*k*ts));
```

```
    yout(k)=0.368*y_1+0.26*y_2+0.1*u_1+0.632*u_2;
    error(k)=rin(k)-yout(k);

%Adjusting Weight Value by hebb learning algorithm
M=4;
if M==1              %No Supervised Heb learning algorithm
   wkp(k)=wkp_1+xiteP*u_1*x(1);  %P
   wki(k)=wki_1+xiteI*u_1*x(2);  %I
   wkd(k)=wkd_1+xiteD*u_1*x(3);  %D
   K=0.06;
elseif M==2           %Supervised Delta learning algorithm
   wkp(k)=wkp_1+xiteP*error(k)*u_1;  %P
   wki(k)=wki_1+xiteI*error(k)*u_1;  %I
   wkd(k)=wkd_1+xiteD*error(k)*u_1;  %D
   K=0.12;
elseif M==3           %Supervised Heb learning algorithm
   wkp(k)=wkp_1+xiteP*error(k)*u_1*x(1);  %P
   wki(k)=wki_1+xiteI*error(k)*u_1*x(2);  %I
   wkd(k)=wkd_1+xiteD*error(k)*u_1*x(3);  %D
   K=0.12;
elseif M==4           %Improved Heb learning algorithm
   wkp(k)=wkp_1+xiteP*error(k)*u_1*(2*error(k)-error_1);
   wki(k)=wki_1+xiteI*error(k)*u_1*(2*error(k)-error_1);
   wkd(k)=wkd_1+xiteD*error(k)*u_1*(2*error(k)-error_1);
   K=0.12;
end

   x(1)=error(k)-error_1;              %P
   x(2)=error(k);                    %I
   x(3)=error(k)-2*error_1+error_2;    %D

   wadd(k)=abs(wkp(k))+abs(wki(k))+abs(wkd(k));
   w11(k)=wkp(k)/wadd(k);
   w22(k)=wki(k)/wadd(k);
   w33(k)=wkd(k)/wadd(k);
   w=[w11(k),w22(k),w33(k)];

    u(k)=u_1+K*w*x;      %Control law

 if u(k)>10
```

```
   u(k)=10;
end
if u(k)<-10
   u(k)=-10;
end

error_2=error_1;
error_1=error(k);

u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);

wkp_1=wkp(k);
wkd_1=wkd(k);
wki_1=wki(k);
end
figure(1);
plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,error,'r');
xlabel('time(s)');ylabel('error');
figure(3);
plot(time,u,'r');
xlabel('time(s)');ylabel('u');
```

4.1.5 基于二次型性能指标学习算法的单神经元自适应PID控制

在最优控制理论中，采用二次型性能指标来计算控制律可以得到所期望的优化效果。在神经元学习算法中，也可借助最优控制中二次型性能指标的思想，在加权系数的调整中引入二次型性能指标，使输出误差和控制增量加权平方和为最小来调整加权系数，从而间接实现对输出误差和控制增量加权的约束控制

设性能指标为：

$$E(k)=\frac{1}{2}(P(\mathrm{rin}(k)-\mathrm{yout}(k))^2+Q\Delta^2 u(k) \tag{4.8}$$

式中，P,Q 分别为输出误差和控制增量的加权系数，$r(k)$ 和 $y(k)$ 为 k 时刻的参考输入和输出。

神经元的输出为：

$$u(k)=u(k-1)+K\sum_{i=1}^{3} w_i(k)x_i(k) \tag{4.9}$$

$$w_i'(k)=w_i(k)/\sum_{i=1}^{3}w_i(k)\qquad (i=1,2,3)$$

$$\begin{aligned}
w_1(k)&=w_1(k-1)+\eta_I K\left[Pb_0 z(k)x_1(k)-QK\sum_{i=1}^{3}(w_i(k)x_i(k))x_1(k)\right]\\
w_2(k)&=w_2(k-1)+\eta_P K\left[Pb_0 z(k)x_2(k)-QK\sum_{i=1}^{3}(w_i(k)x_i(k))x_2(k)\right]\\
w_3(k)&=w_3(k-1)+\eta_D K\left[Pb_0 z(k)x_3(k)-QK\sum_{i=1}^{3}(w_i(k)x_i(k))x_3(k)\right]
\end{aligned}\tag{4.10}$$

式中，b_0 为输出响应的第一个值，且有：

$$\begin{aligned}
x_1(k)&=e(k)\\
x_2(k)&=e(k)-e(k-1)\\
x_3(k)&=\Delta^2 e(k)=e(k)-2e(k-1)+e(k-2)\\
z(k)&=e(k)
\end{aligned}\tag{4.11}$$

4.1.6 仿真程序及分析

仿真实例

设被控制对象过程模型为：

$$\text{yout}(k)=0.368\text{yout}(k-1)+0.264\text{yout}(k-2)+u(k-d)+0.632u(k-d-1)+\xi(k)$$

应用最优二次型性能指标学习算法进行仿真研究。$\xi(k)$ 为在 100 个采样时间的外加干扰，$\xi(100)=0.10$，输入为阶跃响应信号 $\text{rin}(k)=1.0$。启动时采用开环控制，取 $u=0.1726, K=0.02, P=2, Q=1, d=6$，比例、积分、微分三部分加权系数学习速率分别取 $\eta_I=4, \eta_P=120, \eta_D=159$，$w_1(0)=0.34, w_2(0)=0.32, w_3(0)=0.33$，神经元自适应 PID 位置跟踪及控制过程中权值变化结果如图 4-6 和图 4-7 所示。

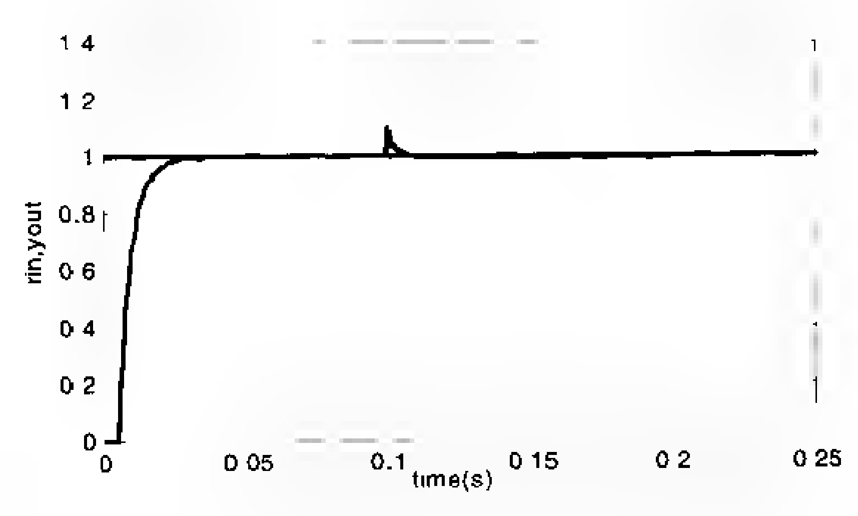


图 4-6 二次型性能指标学习单神经元自适应 PID 位置跟踪

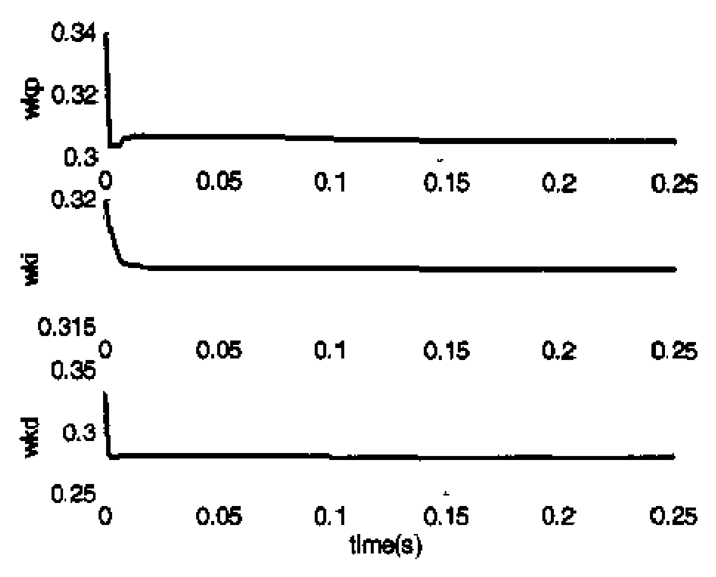


图 4-7　单神经元 PID 控制过程中权值变化

仿真程序：chap4_2.m。

```
%Single Neural Net PID Controller based on Second Type Learning Algorithm
clear all;
close all;

xc=[0,0,0]';

K=0.02;P=2;Q=1;d=6;

xiteP=120;
xiteI=4;
xiteD=159;

%Initilizing kp,ki and kd
wkp_1=rand;
wki_1=rand;
wkd_1=rand;

wkp_1=0.34;
wki_1=0.32;
wkd_1=0.33;

error_1=0;error_2=0;
y_1=0;y_2=0;
u_1=0.1726;u_2=0;u_3=0;u_4=0;u_5=0;u_6=0;u_7=0;
```

```
ts=0.001;
for k=1:1:250
    time(k)=k*ts;
    rin(k)=1.0;                          %Tracing Step Signal

ym(k)=0;
if k==100
   ym(k)=0.10;  %Disturbance
end
yout(k)=0.368*y_1+0.26*y_2+u_6+0.632*u_7+ym(k);
error(k)=rin(k)-yout(k);

wx=[wkp_1,wkd_1,wki_1];
wx=wx*xc;

b0=yout(1);
K=0.0175;
wkp(k)=wkp_1+xiteP*K*[P*b0*error(k)*xc(1)-Q*K*wx*xc(1)];
wki(k)=wki_1+xiteI*K*[P*b0*error(k)*xc(2)-Q*K*wx*xc(2)];
wkd(k)=wkd_1+xiteD*K*[P*b0*error(k)*xc(3)-Q*K*wx*xc(3)];

   xc(1)=error(k)-error_1;              %P
   xc(2)=error(k);                      %I
   xc(3)=error(k)-2*error_1+error_2;    %D

   wadd(k)=abs(wkp(k))+abs(wki(k))+abs(wkd(k));
   w11(k)=wkp(k)/wadd(k);
   w22(k)=wki(k)/wadd(k);
   w33(k)=wkd(k)/wadd(k);
   w=[w11(k),w22(k),w33(k)];

u(k)=u_1+K*w*xc;     % Control law

if u(k)>=10
   u(k)=10;
end
if u(k)<=-10
   u(k)=-10;
end

error_2=error_1;
error_1=error(k);
```

```
. 7 ._6;u 6 u 5;._5 . 4;u 4 . 3,
u 3 . 2;._2 u  , . 1 u(k ;

wkp_1 wkp(k ;
wkd_1 wkd(k ;
wki_1 wki k ;

y_2 y_1;y_1 yout(k ;
end
figure 1,;
plot time,rin,'r ,time,yout, b',;
xlabel 't.me(s ',;ylabel('rin,yout );
figure 2);
plot t.me,.,'r' ;
xlabel 'time(s   ;ylabel('.  ;
figure(3);
subplot(311 ,
plot time,wkp,'r  ,
xlabel('time(s ' ;ylabel('wkp  ;
subpl t 312 ·
plot t.me,wki,'r  ,
xlabel 'time s ' ;ylabel('wki ,,
subplot 31( ;
plot time,wkd,'r  ,
xlabel('time(s ');ylabel('wkd );
```

4.2 基于 BP 神经网络整定的 PID 控制

4.2.1 基于 BP 神经网络的 PID 整定原理

PID 控制要取得较好的控制效果，就必须通过调整好比例、积分和微分三种控制作用，形成控制量中既相互配合又相互制约的关系，这种关系不一定是简单的“线性组合”，从变化无穷的非线性组合中可以找出最佳的。神经网络所具有的任意非线性表达能力，可以通过对系统性能的学习来实现具有最佳组合的 PID 控制。采用 BP 网络，可以建立参数 k_p，k_i，k_d 自学习的 PID 控制器。

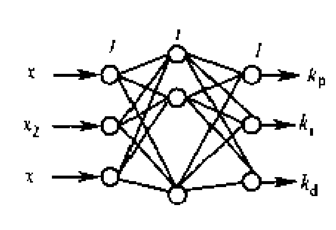


图 4-8　BP 网络结构

基于 BP（Back Propagation）网络的 PID 控制系统结构如图 4-8 所示，控制器由两部分构成：

（1）经典的 PID 控制器，直接对被控对象进行闭环控制，并且三个参数 k_p，k_i，k_d 为在线调整方式。

（2）神经网络，根据系统的运行状态，调节 PID 控制器的参数，以期达到某种性能指标的最优化，使输出层神经元的输出状态对应于 PID 控制器的三个可调参数 k_p，k_i，k_d 通过神经网络的自学习、加权系数调整，使神经网络输出对应于某种最优控制律下的 PID 控制器参数。

经典增量式数字 PID 的控制算法为：

$$u(k)=u(k-1)+\Delta u(k)$$

$$\Delta u(k)=k_p(\mathrm{error}(k)-\mathrm{error}(k-1))+k_i\mathrm{error}(k)+k_d(\mathrm{error}(k)-2\mathrm{error}(k-1)+\mathrm{error}(k-2)) \tag{4.12}$$

式中，k_p，k_i，k_d 分别为比例、积分、微分系数。

采用三层 BP 网络，其结构如图 4-8 所示。

网络输入层的输入为：

$$O_j^{(1)}=x(j) \qquad (j=1,2,\cdots,M) \tag{4.13}$$

式中，输入变量的个数 M 取决于被控系统的复杂程度。

网络隐含层的输入、输出为：

$$\begin{aligned}\mathrm{net}_i^{(2)}(k)&=\sum_{j=0}^{M}w_{ij}^{(2)}O_j^{(1)}\\ O_i^{(2)}(k)&=f(\mathrm{net}_i^{(2)}(k)) \qquad (i=1,2,\cdots,Q)\end{aligned} \tag{4.14}$$

式中，$w_{ij}^{(2)}$ 为隐含层加权系数；上角标（1）、（2）、（3）分别代表输入层、隐含层和输出层。

隐层神经元的活化函数取正负对称的 Sigmoid 函数：

$$f(x)=\tanh(x)=\frac{e^x-e^{-x}}{e^x+e^{-x}} \tag{4.15}$$

网络输出层的输入输出为：

$$\begin{aligned}\mathrm{net}_l^{(3)}(k)&=\sum_{i=0}^{Q}w_{li}^{(3)}O_i^{(2)}(k)\\ O_l^{(3)}(k)&=g(\mathrm{net}_l^{(3)}(k)) \qquad (l=1,2,3)\\ O_1^{(3)}(k)&=k_p\\ O_2^{(3)}(k)&=k_i\\ O_3^{(3)}(k)&=k_d\end{aligned} \tag{4.16}$$

输出层输出节点分别对应三个可调参数 k_p，k_i，k_d。由于 k_p，k_i，k_d 不能为负值，所以输出层神经元的活化函数取非负的 Sigmoid 函数：

$$g(x)=\frac{1}{2}(1+\tanh(x))=\frac{e^x}{e^x+e^{-x}} \tag{4.17}$$

取性能指标函数为：

$$E(k)=\frac{1}{2}(\mathrm{rin}(k)-\mathrm{yout}(k))^2 \tag{4.18}$$

按照梯度下降法修正网络的权系数，即按 $E(k)$ 对加权系数的负梯度方向搜索调整，并附加一个使搜索快速收敛全局极小的惯性项：

$$\Delta w_{li}^{(3)}(k)=-\eta\frac{\partial E(k)}{\partial w_{li}^{(3)}}+\alpha\Delta w_{li}^{(3)}(k-1) \tag{4.19}$$

式中，η 为学习速率；α 为惯性系数。

$$\frac{\partial E(k)}{\partial w_{li}^{(3)}}=\frac{\partial E(k)}{\partial y(k)}\cdot\frac{\partial y(k)}{\partial \Delta u(k)}\cdot\frac{\partial \Delta u(k)}{\partial O_l^{(3)}(k)}\cdot\frac{\partial O_l^{(3)}(k)}{\partial \text{net}_l^{(3)}(k)}\cdot\frac{\partial \text{net}_l^{(3)}(k)}{\partial w_{li}^{(3)}(k)} \tag{4.20}$$

$$\frac{\partial \text{net}_l^{(3)}(k)}{\partial w_{li}^{(3)}(k)}=O_i^{(2)}(k) \tag{4.21}$$

由于 $\frac{\partial y(k)}{\partial \Delta u(k)}$ 未知，所以近似用符号函数 $\text{sgn}\left(\frac{\partial y(k)}{\partial \Delta u(k)}\right)$ 取代，由此带来计算不精确的影响可以通过调整学习速率 η 来补偿。

由式（4.12）和式（4.16），可求得：

$$\frac{\partial \Delta u(k)}{\partial O_1^{(3)}(k)}=\text{error}(k)-\text{error}(k-1) \tag{4.22}$$

$$\frac{\partial \Delta u(k)}{\partial O_2^{(3)}(k)}=\text{error}(k) \tag{4.23}$$

$$\frac{\partial \Delta u(k)}{\partial O_3^{(3)}(k)}=\text{error}(k)-2\text{error}(k-1)+\text{error}(k-2) \tag{4.24}$$

上述分析可得网络输出层权的学习算法为：

$$\Delta w_{li}^{(3)}(k)=\alpha\Delta w_{li}^{(3)}(k-1)+\eta\delta_l^{(3)}O_i^{(2)}(k) \tag{4.25}$$

$$\delta_l^{(3)}=\text{error}(k)\text{sgn}\left(\frac{\partial y(k)}{\partial \Delta u(k)}\right)\frac{\partial \Delta u(k)}{\partial O_l^{(3)}(k)}g'(\text{net}_l^{(3)}(k)) \qquad (l=1,2,3) \tag{4.26}$$

同理可得隐含层加权系数的学习算法：

$$\Delta w_{ij}^{(2)}(k)=\alpha\Delta w_{ij}^{(2)}(k-1)+\eta\delta_i^{(2)}O_j^{(1)}(k) \tag{4.27}$$

$$\delta_i^{(2)}=f'(\text{net}_i^{(2)}(k))\sum_{l=1}^{3}\delta_l^{(3)}w_{li}^{(3)}(k) \qquad (i=1,2,\cdots,Q) \tag{4.28}$$

式中，$g'(\cdot)=g(x)(1-g(x))$，$f'(\cdot)=(1-f^2(x))/2$。（4.29）

基于BP网络的PID控制器结构如图4-9所示，该控制器控制算法归纳如下：

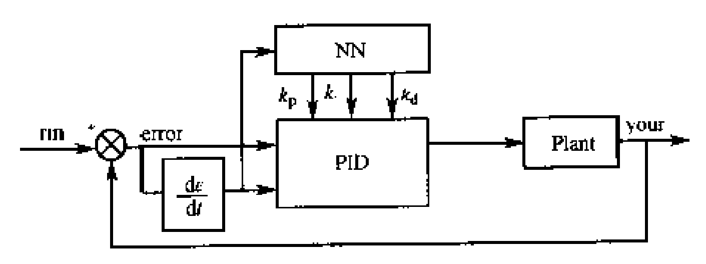


图4-9　基于BP网络的PID控制器结构

（1）确定BP网络的结构，即确定输入层节点数 M 和隐含层节点数 Q，并给出各层加权系数的初值 $w_{ij}^{(1)}(0)$ 和 $w_{li}^{(2)}(0)$，选定学习速率 η 和惯性系数 α，此时 $k=1$；

（2）采样得到 $\text{rin}(k)$ 和 $\text{yout}(k)$，计算该时刻误差 $\text{error}(k)=\text{rin}(k)-\text{yout}(k)$；

（3）计算神经网络NN各层神经元的输入、输出，NN输出层的输出即为PID控制器的三个可调参数 k_p，k_i，k_d；

（4）根据式(4.12)计算PID控制器的输出 $u(k)$；

（5）进行神经网络学习，在线调整加权系数 $w_{ij}^{(1)}(k)$ 和 $w_{li}^{(2)}(k)$，实现PID控制参数的自适应调整；

（6）置$k=k+1$，返回到（1）。

4.2.2 仿真程序及分析

仿真实例

设被控制对象的近似数学模型为：

$$yout(k)=\frac{a(k)yout(k-1)}{1+yout^2(k-1)}+u(k-1)$$

式中，系数$a(k)$是慢时变的，$a(k)=1.2(1-0.8e^{-0.1k})$。

神经网络的结构选择 4-5-3，学习速率$\eta=0.28$和惯性系数$\alpha=0.04$，加权系数初始值取区间$[-0.5,0.5]$上的随机数。输入指令信号分为两种：

（1）$rin(k)=1.0$；

（2）$rin(k)=\sin(2\pi t)$。取$S=1$时为阶跃跟踪，$S=2$时为正弦跟踪，初始权值取随机值，运行稳定后用稳定权值代替随机值。其跟踪结果和相应的曲线如图 4-10～图 4-14 所示。

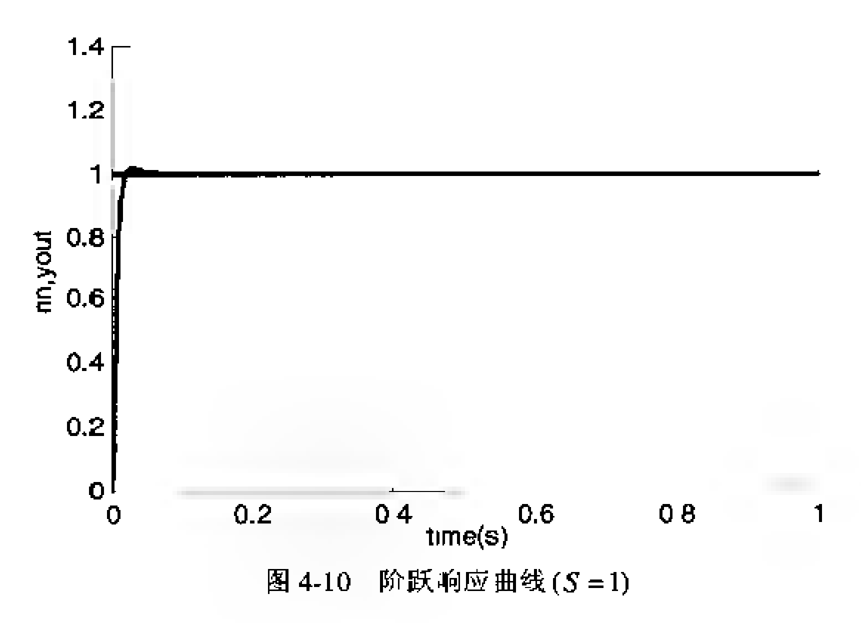


图 4-10　阶跃响应曲线（$S=1$）

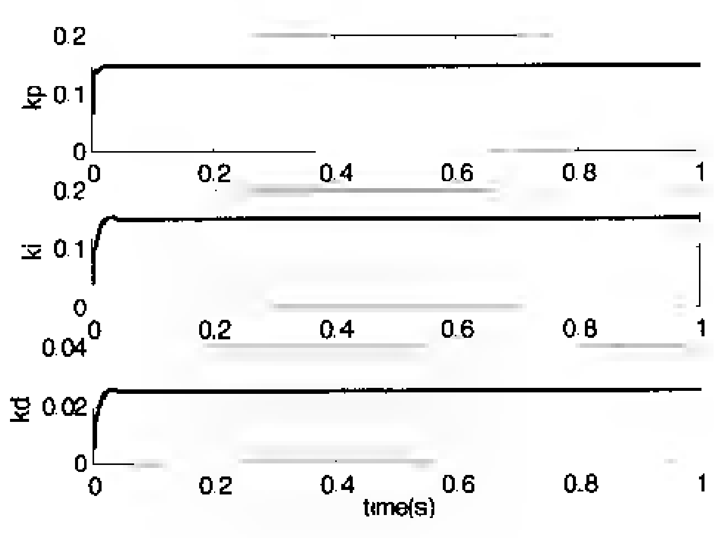


图 4-11　参数自适应整定曲线

由于可调参数k_p，k_i，k_d均取非负的 Sigmoid 函数，其值在(0, 1)之间，使得本算法的应

用具有局限性，读者可根据需要进行改进。

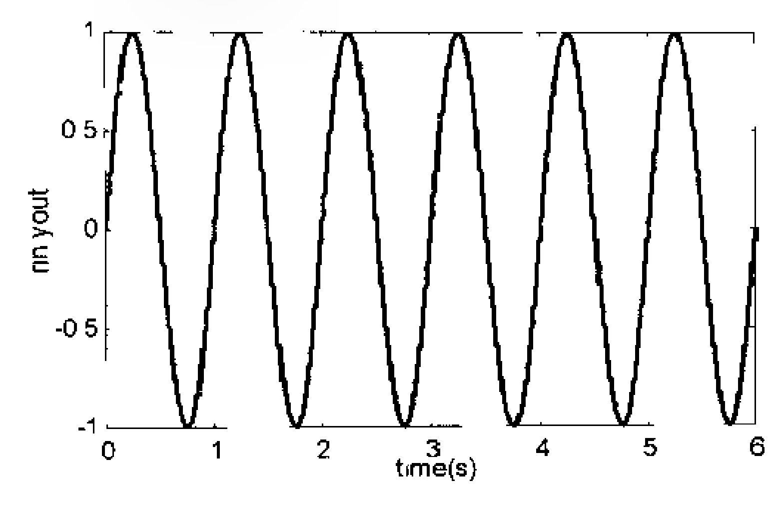


图 4-12　正弦跟踪曲线（S=2）

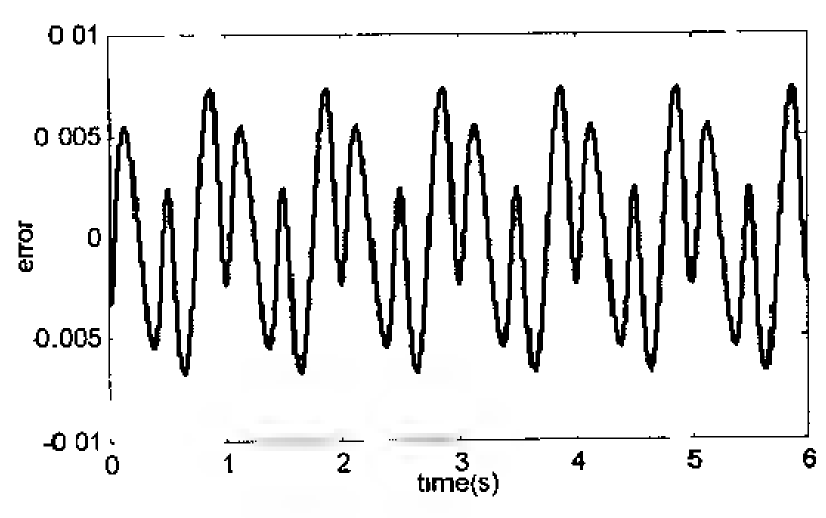


图 4-13　跟踪误差曲线

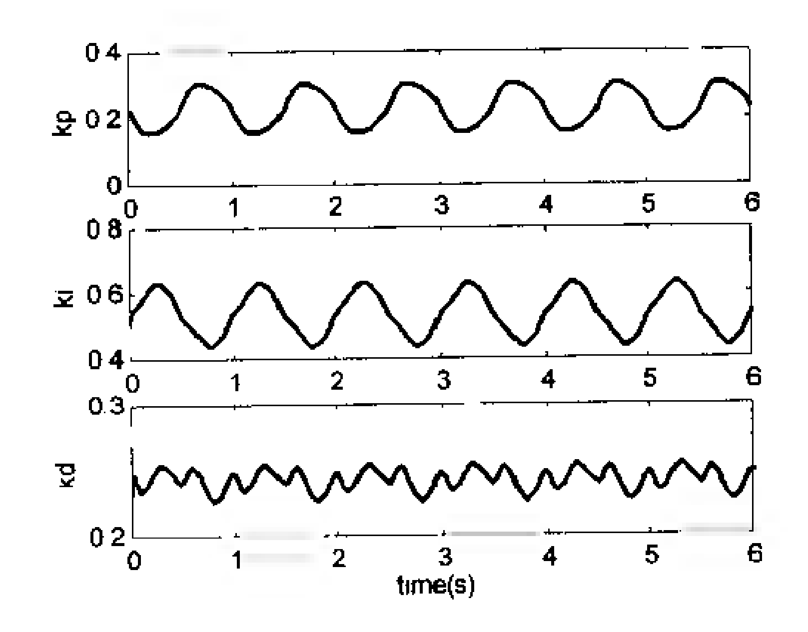


图 4-14　参数自适应整定曲线

仿真程序：chap4_3.m。

```
%BP based PID Control
clear all;
close all;

xite 0.20;
alfa 0.05;

S 2; %Signal type

IN 4;H 5,Out 3;  %NN Structure
if S 1  %Step Signal
w1   0.6394    [illegible].2696    0.[illegible]756    0.7023;
     0.8603    .2[illegible]1[illegible]    0.5024    0.2596;
     1.0749    0.[illegible]543    1.6820    0.5437;
     0.3625    0.0724    0 6463    0 2859;
     0.142[illegible]    0.0279    0.5406    [illegible].7660];
%w1 0 50*rands(H,IN);
w1_1-w1;w[illegible]_2-w1;w1_3-w1;
wo [illegible]0.7576 0 2616 [illegible].5820  0.1416  0.1[illegible]25;
   -0.1146 0.2949 0 8352  0.2205  0.4508;
    [illegible].7201 0.4566 0.7672  0.4962  0.3632[illegible];
%wo 0.50*rands Out,H ;
wo_1-wo;wo_2-wo;wo_3-wo;
end

if S 2  %Sine Signal
w1 [ 0.2846    0.2193    0.5097   -1.0668;
     0.7484   -0.1210    0.4708    0.0988;
    -0.7176    0.8297    1.6000    0.2049;
    -0.0858    0.1925    0.6[illegible]46    0.0347;
     0.4358    0.2369    0.4564    0.1324];
%w1 0.50*rands(H,IN),
w1 1 w1;w1 2-w1;w1 3-w1;
wo [1.04[illegible]8    0 5478    0.8682    0.1446    0.15[illegible]7;
    0.1716    0.5811    1.1214    0.5067    0.7370;
    1.006[illegible]    0.7428    1.0534    0.7824    0.6494];
%wo 0 50*rands Out,H ;
wo_1 wo,wo_2 wo;wo 3 wo;
end
```

```
x [0,0,0],
du_1 0;
u_1 0;u 2-0;u_3 0;u_4-0;u_5 0;
y_1-0;y_2 0;y_3 0;

Oh zeros H,1);    %Output from NN middle layer
I Oh;             %Input to NN middle layer
error_2 0;
error 1-0,

ts 0.001;
for k 1:1:6000
time k) k*ts,

if S 1
  rin(k 1 0;
elseif S- 2
  rin(k -sin(1*2*pi*k*ts);
end

%Unlinear model
a(k  1.2* 1-0.8*exp( 0.1*k));
yout(k) a k *y 1 (1+y 1^2)+u 1;

error(k -rin k) yout(k);

xi [rin k),yout(k ,error(k),1];

x(1)-error k  error_1;
x(2) error(k),
x(3) error(k) 2*error 1+error_2;

epid [x(1 ;x(2 ;x(3,];
I xi*wi ;
for j 1:1:H
   Oh(j) - exp(I j))-exp(-I(j)))/ exp(I j))+exp( I j)) ; %Middle Layer
end
K wo*Oh;              %Output Layer
for l-1:1:Out
   K(l  exp K(l )/(exp(K(l),+exp( K(l)));        %Getting kp,ki,kd
end
kp k) K 1 ;ki(k) K(2 ;kd(k) K(3);
```

```
Kpid=[kp(k),ki(k),kd(k)];

du(k)=Kpid*epid;
u(k)=u_1+du(k);

dyu(k)=sign((yout(k)-y_1)/(du(k)-du_1+0.0001));

%Output layer
for j=1:1:Out
    dK(j)=2/(exp(K(j))+exp(-K(j)))^2;
end
for l=1:1:Out
    delta3(l)=error(k)*dyu(k)*epid(l)*dK(l);
end

for l=1:1:Out
   for i=1:1:H
       d_wo=xite*delta3(l)*Oh(i)+alfa*(wo_1-wo_2);
   end
end
    wo=wo_1+d_wo+alfa*(wo_1-wo_2);
%Hidden layer
for i=1:1:H
    dO(i)=4/(exp(I(i))+exp(-I(i)))^2;
end
    segma=delta3*wo;
for i=1:1:H
   delta2(i)=dO(i)*segma(i);
end

d_wi=xite*delta2'*xi;
wi=wi_1+d_wi+alfa*(wi_1-wi_2);

%Parameters Update
du_1=du(k);
u_5=u_4;u_4=u_3;u_3=u_2;u_2=u_1;u_1=u(k);
y_2=y_1;y_1=yout(k);

wo_3=wo_2;
wo_2=wo_1;
wo_1=wo;
```

```
w1_3=w1_2;
w1_2=w1_1;
w1_1=w1;

error_2=error_1;
error_1=error(k);
end
figure(1);
plot(time,rin,'r',time,yout,'b');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,error,'r');
xlabel('time(s)');ylabel('error');
figure(3);
plot(time,u,'r');
xlabel('time(s)');ylabel('u');
figure(4);
subplot(311);
plot(time,kp,'r');
xlabel('time(s)');ylabel('kp');
subplot(312);
plot(time,ki,'g');
xlabel('time(s)');ylabel('ki');
subplot(313);
plot(time,kd,'b');
xlabel('time(s)');ylabel('kd');
```

4.3 基于 RBF 神经网络整定的 PID 控制

4.3.1 RBF 神经网络模型

径向基函数（Radial Basis Function，RBF）神经网络是由 J. Moody 和 C. Darken 在 20 世纪 80 年代末提出的一种神经网络，它是具有单隐层的三层前馈网络。由于它模拟了人脑中局部调整、相互覆盖接收域（或称感受野，Receptive Field）的神经网络结构，因此，RBF 网络是一种局部逼近网络，已证明它能以任意精度逼近任意连续函数。

1. 网络结构

RBF 网络是一种三层前向网络，由输入到输出的映射是非线性的，而隐含层空间到输出空间的映射是线性的，从而大大加快了学习速度并避免局部极小问题。

RBF 神经网络结构如图 4-15 所示。

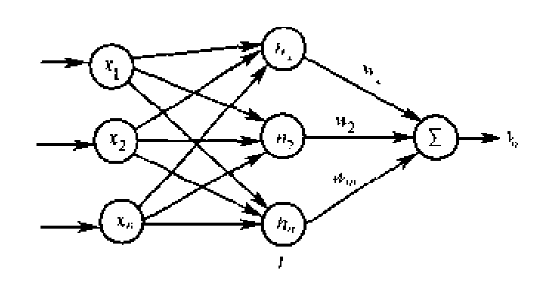


图 4-15 RBF 神经网络结构

2 被控对象 Jacobian 信息的辨识算法

在 RBF 网络结构中，$\boldsymbol{X}=[x_1, x_2, \cdots, x_n]^{\mathrm{T}}$ 为网络的输入向量。设 RBF 网络的径向基向量 $\boldsymbol{H}=[h_1, h_2, \cdots, h_j, \cdots, h_m]^{\mathrm{T}}$，其中 h_j 为高斯基函数：

$$h_j=\exp\left(-\frac{\|\boldsymbol{X}-\boldsymbol{C}_j\|^2}{2b_j^2}\right) \quad (j=1,2,\cdots,m) \tag{4.30}$$

网络的第 j 个结点的中心矢量为 $\boldsymbol{C}_j=[c_{j1}, c_{j2}, \cdots, c_{ji}, \cdots, c_{jn}]^{\mathrm{T}}$，其中，$i=1, 2, \cdots, n$

设网络的基宽向量为：

$$\boldsymbol{B}=[b_1, b_2, \cdots, b_m]^{\mathrm{T}}$$

b_j 为节点 j 的基宽度参数，且为大于零的数。网络的权向量为：

$$\boldsymbol{W}=[w_1, w_2, \cdots, w_j, \cdots, w_m]^{\mathrm{T}} \tag{4.31}$$

辨识网络的输出为：

$$y_m(k)=w_1h_1+w_2h_2+\cdots+w_mh_m \tag{4.32}$$

辨识器的性能指标函数为：

$$J_1=\frac{1}{2}(\mathrm{yout}(k)-y_m(k))^2 \tag{4.33}$$

根据梯度下降法，输出权、节点中心及节点基宽参数的迭代算法如下：

$$w_j(k)=w_j(k-1)+\eta(\mathrm{yout}(k)-y_m(k))h_j+\alpha(w_j(k-1)-w_j(k-2)) \tag{4.34}$$

$$\Delta b_j=(\mathrm{yout}(k)-y_m(k))w_jh_j\frac{\|\boldsymbol{X}-\boldsymbol{C}_j\|^2}{b_j^3} \tag{4.35}$$

$$b_j(k)=b_j(k-1)+\eta\Delta b_j+\alpha(b_j(k-1)-b_j(k-2)) \tag{4.36}$$

$$\Delta c_{ji}=(\mathrm{yout}(k)-y_m(k))w_j\frac{x_j-c_{ji}}{b_j^2} \tag{4.37}$$

$$c_{ji}(k)=c_{ji}(k-1)+\eta\Delta c_{ji}+\alpha(c_{ji}(k-1)-c_{ji}(k-2)) \tag{4.38}$$

式中，η 为学习速率，α 为动量因子，

Jacobian 阵（即为对象的输出对控制输入变化的灵敏度信息）算法为：

$$\frac{\partial y(k)}{\partial \Delta u(k)}\approx\frac{\partial y_m(k)}{\partial \Delta u(k)}=\sum_{j=1}^{m}w_jh_j\frac{c_{ji}-x_1}{b_j^2} \tag{4.39}$$

式中，$x_1=\Delta u(k)$。

4.3.2 RBF 网络 PID 整定原理

采用增量式 PID 控制器，控制误差为：

$$\text{error}(k)=\text{rin}(k)-\text{yout}(k)$$

PID 三项输入为：

$$
\begin{aligned}
xc(1)&=\text{error}(k)-\text{error}(k-1)\\
xc(2)&=\text{error}(k)\\
xc(3)&=\text{error}(k)-2\text{error}(k-1)+\text{error}(k-2)
\end{aligned}
\tag{4.40}
$$

控制算法为：

$$u(k)=u(k-1)+\Delta u(k)$$

$$\Delta u(k)=k_{\mathrm{p}}(\text{error}(k)-\text{error}(k-1))+k_{\mathrm{i}}\text{error}(k)+k_{\mathrm{d}}(\text{error}(k)-2\text{error}(k-1)+\text{error}(k-2)) \tag{4.41}$$

神经网络整定指标为：

$$E(k)=\frac{1}{2}\text{error}(k)^2 \tag{4.42}$$

$k_{\mathrm{p}},k_{\mathrm{i}},k_{\mathrm{d}}$ 的调整采用梯度下降法：

$$\Delta k_{\mathrm{p}}=-\eta\frac{\partial E}{\partial k_{\mathrm{p}}}=-\eta\frac{\partial E}{\partial y}\frac{\partial y}{\partial \Delta u}\frac{\partial \Delta u}{\partial k_{\mathrm{p}}}=\eta\text{error}(k)\frac{\partial y}{\partial \Delta u}xc(1) \tag{4.43}$$

$$\Delta k_{\mathrm{i}}=-\eta\frac{\partial E}{\partial k_{\mathrm{i}}}=-\eta\frac{\partial E}{\partial y}\frac{\partial y}{\partial \Delta u}\frac{\partial \Delta u}{\partial k_{\mathrm{i}}}=\eta\text{error}(k)\frac{\partial y}{\partial \Delta u}xc(2) \tag{4.44}$$

$$\Delta k_{\mathrm{d}}=-\eta\frac{\partial E}{\partial k_{\mathrm{d}}}=-\eta\frac{\partial E}{\partial y}\frac{\partial y}{\partial \Delta u}\frac{\partial \Delta u}{\partial k_{\mathrm{d}}}=\eta\text{error}(k)\frac{\partial y}{\partial \Delta u}xc(3) \tag{4.45}$$

式中，$\dfrac{\partial y}{\partial \Delta u}$ 为被控对象的 Jacobian 信息，可通过神经网络的辨识而得。

RBF 整定 PID 控制系统的结构如图 4-16 所示。

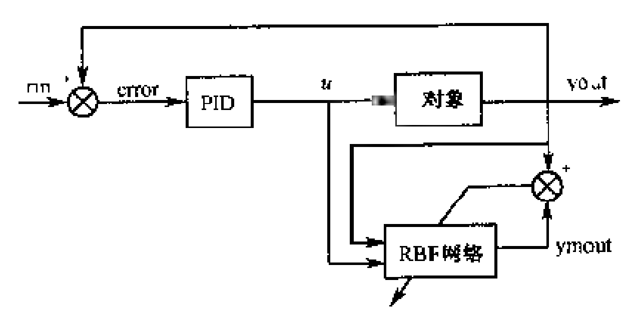


图 4-16 RBF 网络整定 PID 控制框图

4.3.3 仿真程序及分析

仿真实例

设被控制对象为：

$$\text{yout}(k)=\frac{-0.1\text{yout}(k-1)+u(k-1)}{1+\text{yout}(k-1)^2}$$

输入指令信号为 $\text{rin}(t)=1.0\text{sgn}(\sin(2\pi t))$，RBF 网络结构选为 3-6-1，网络辨识的三个输

入为：$\Delta u(k), yout(k), yout(k-1)$。$M=1$ 时为 RBF 整定的 PID 控制，其结果如图 4-17～图 4-19 所示，$M=2$ 时为未加整定的 PID 控制，其结果如图 4-20 所示。

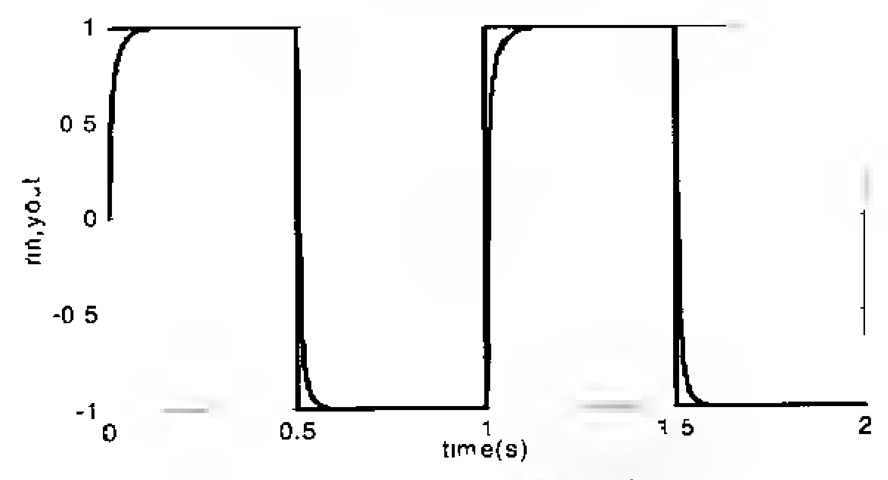


图 4-17 RBF 整定 PID 控制方波响应（M=1）

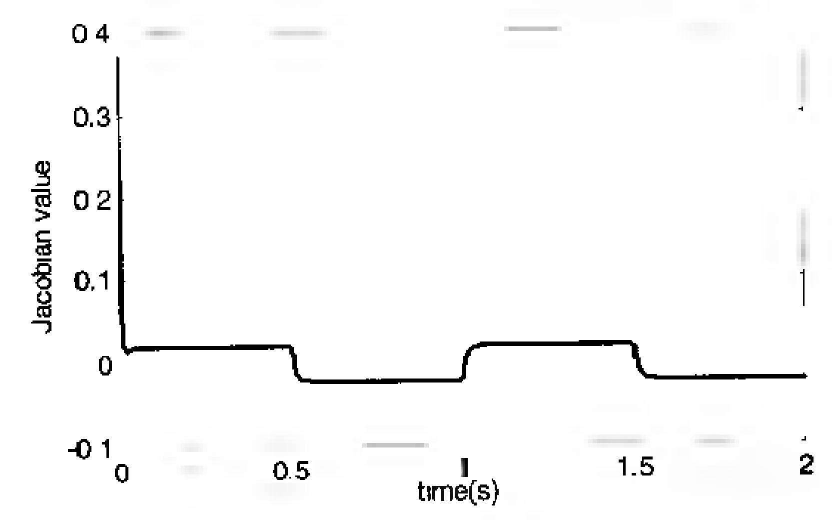


图 4-18 对象 Jacobian 信息的辨识结果

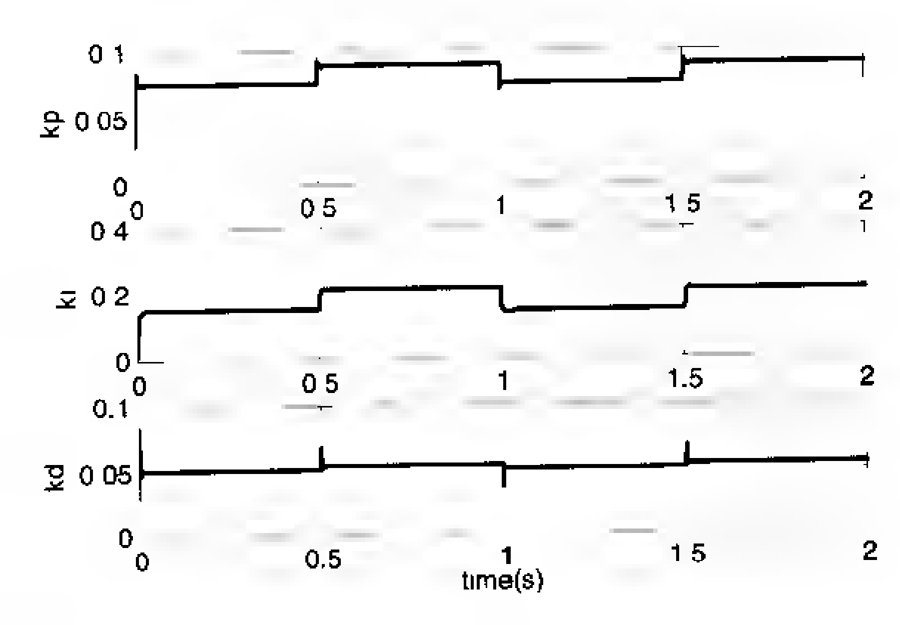


图 4-19 参数自适应整定曲线

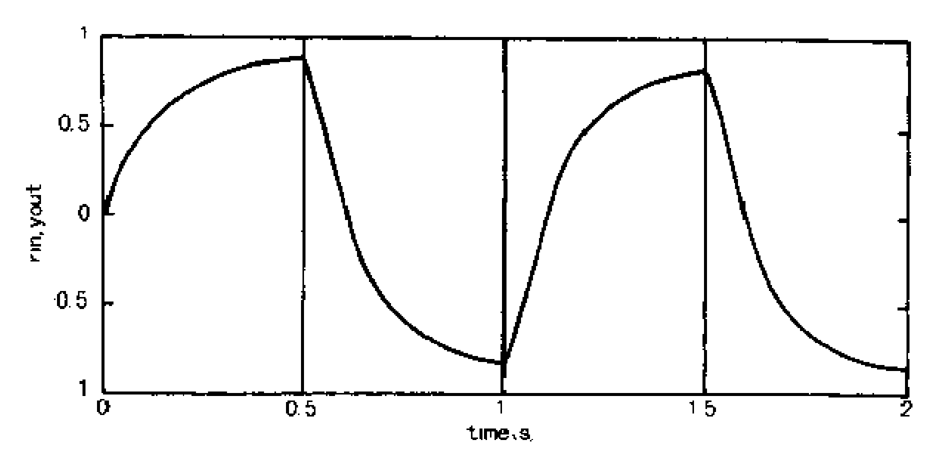


图 4-20　未整定的 PID 控制方波响应（M=2）

仿真程序：chap4_4.m。

```
%Adaptive PID control based on RBF Identification
clear all;
close all;

xite 0.25;
alfa 0.05;
belte 0.01;
x , , ,0]';

ci 30*ones(3,6 ;
bi 40*ones 6,1),
w 10*ones 6,1);

h [0,0,0,0,0,0]';

ci_1 ci;ci 3-ci 1;ci 2 ci_1;
bi 1 bi;bi_2 bi 1;bi 3-bi 2;
w 1 w;w 2-w_1;w_3 w 1;

u_1-0;y_1 0;
xc-[0,0,0]';
error_1 0,error 2 0;error 0;

%kp rand 1 ;
%ki-rand( ;
%kd-rand(1 ;
kp0 0.03;
ki0-0.01;
```

```
kd0=0.0;

kp_1=kp0;
kd_1=kd0;
ki_1=ki0;

xitekp=0.20;
xitekd=0.20;
xiteki=0.20;

ts=0.001;
for k=1:1:2000
   time(k)=k*ts;
   rin(k)=1.0*sign(sin(2*pi*k*ts));
   yout(k)=(-0.1*y_1+u_1)/(1+y_1^2);  %Nonlinear plant

   for j=1:1:6
      h(j)=exp(-norm(x-ci(:,j))^2/(2*bi(j)*bi(j)));
   end
   ymout(k)=w'*h;

   d_w=0*w;
   for j=1:1:6
      d_w(j)=xite*(yout(k)-ymout(k))*h(j);
   end
   w=w_1+d_w+alfa*(w_1-w_2)+belte*(w_2-w_3);

   d_bi=0*bi;
   for j=1:1:6
d_bi(j)=xite*(yout(k)-ymout(k))*w(j)*h(j)*(bi(j)^-3)*norm(x-ci(:,j))^2;
   end
   bi=bi_1+ d_bi+alfa*(bi_1-bi_2)+belte*(bi_2-bi_3);
   for j=1:1:6
     for i=1:1:3
d_ci(i,j)=xite*(yout(k)-ymout(k))*w(j)*h(j)*(x(i)-ci(i,j))*(bi(j)^-2);
     end
   end
   ci=ci_1+d_ci+alfa*(ci_1-ci_2)+belte*(ci_2-ci_3);
```

```
%%%%%%%%%%%%%%%%%%%%%%Jacobian%%%%%%%%%%%%%%%%%%%%%%
  yu=0;
  for j=1:1:6
      yu=yu+w(j)*h(j)*(-x(1)+ci(1,j))/bi(j)^2;
  end
  dyout(k)=yu;
%%%%%%%%%%%%%%%%%%%%%%Start of Control system%%%%%%%%%%%%%%%%%%
   error(k)=rin(k)-yout(k);
   kp(k)=kp_1+xitekp*error(k)*dyout(k)*xc(1);
   kd(k)=kd_1+xitekd*error(k)*dyout(k)*xc(2);
   ki(k)=ki_1+xiteki*error(k)*dyout(k)*xc(3);
   if kp(k)<0
      kp(k)=0;
   end
   if kd(k)<0
      kd(k)=0;
   end
   if ki(k)<0
      ki(k)=0;
   end

        M=1;
        switch M
        case 1
        case 2  %Only PID Control
         kp(k)=kp0;
         ki(k)=ki0;
         kd(k)=kd0;
   end
     du(k)=kp(k)*xc(1)+kd(k)*xc(2)+ki(k)*xc(3);
   u(k)=u_1+du(k);

%Return of parameters
   x(1)=du(k);
   x(2)=yout(k);
   x(3)=y_1;


     u_1=u(k);
```

```
    y_1=yout(k);

    ci_3=ci_2;
    ci_2=ci_1;
    ci_1=ci;

    bi_3=bi_2;
    bi_2=bi_1;
    bi_1=bi;

    w_3=w_2;
    w_2=w_1;
    w_1=w;

    xc(1)=error(k)-error_1;             %Calculating P
    xc(2)=error(k)-2*error_1+error_2;   %Calculating D
    xc(3)=error(k);                     %Calculating I

    error_2=error_1;
    error_1=error(k);

    kp_1=kp(k);
    kd_1=kd(k);
    ki_1=ki(k);
end
figure(1);
plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,yout,'r',time,ymout,'b');
xlabel('time(s)');ylabel('yout,ymout');
figure(3);
plot(time,dyout);
xlabel('time(s)');ylabel('Jacobian value');
figure(4);
subplot(311);
plot(time,kp,'r');
xlabel('time(s)');ylabel('kp');
subplot(312);
plot(time,ki,'r');
```

```
xlabel('time(s)');ylabel('ki');
subplot(313);
plot(time,kd,'r');
xlabel('time(s)');ylabel('kd');
```

4.4 基于 RBF 神经网络辨识的单神经元 PID 模型参考自适应控制

4.4.1 神经网络模型参考自适应控制原理

图 4-21 示出单神经元 PID 模型参考自适应控制系统框图，单神经元 PID 作为控制器 NNC，RBF 网络作为辨识器 NNI，实现对被控对象的 Jacobian 信息辨识。

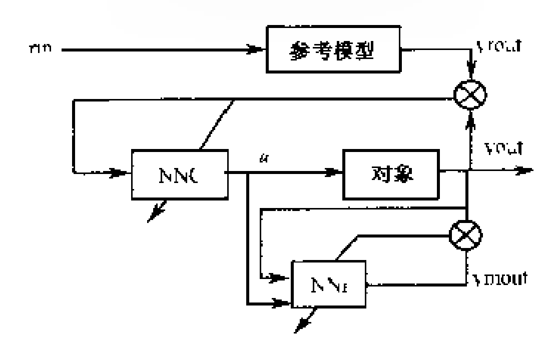


图 4-21　单神经元 PID 模型参考自适应控制系统框图

单神经元网络的输入为：

$$
\begin{aligned}
xc_1(k) &= \mathrm{error}(k) \\
xc_2(k) &= (\mathrm{error}(k)-\mathrm{error}(k-1))/T \\
xc_3(k) &= \sum_{k=1}^{k}\mathrm{error}(k)T
\end{aligned}
\tag{4.46}
$$

神经网络的输出即为控制器的输出：

$$
u(k)=\sum_{j=1}^{3} wc_j(k)xc_j(k) \tag{4.47}
$$

神经网络控制器的学习算法采用 delta 学习规则：

$$
E=\frac{1}{2}\mathrm{error}(k)^2=\frac{1}{2}(\mathrm{yrout}(k)-\mathrm{yout}(k))^2 \tag{4.48}
$$

$$
\Delta wc_j(k)=-\eta\frac{\partial E}{\partial wc_j}=-\eta\frac{\partial E}{\partial y}\frac{\partial y}{\partial u}\frac{\partial u}{\partial wc_j}=\eta\mathrm{error}(k)\frac{\partial y}{\partial u}xc_j(k) \tag{4.49}
$$

式中，$\frac{\partial y}{\partial u}$ 为被控对象的 Jacobian 信息，通过 RBF 神经网络的辨识而得。

4.4.2 仿真程序及分析

仿真实例

针对二阶传递函数进行单神经元 PID 模型参考自适应控制，被控对象为：

$$
G_p(s)=\frac{1}{0.003s^2+0.067s}
$$

采样时间为 1ms，$S=1$时，输入指令信号为正弦信号；$S=2$时，输入指令信号为参考模型。取$S=2$，参数辨识、Jacobian 信息辨识及正弦位置跟踪结果如图 4-22～图 4-24 所示。

参考模型指令信号为：

$$\text{rin}(k)=0.50\sin(0.006\pi k)$$

$$\text{yrout}(k)=0.2\text{yrout}(k-1)+0.6\text{rin}(k)$$

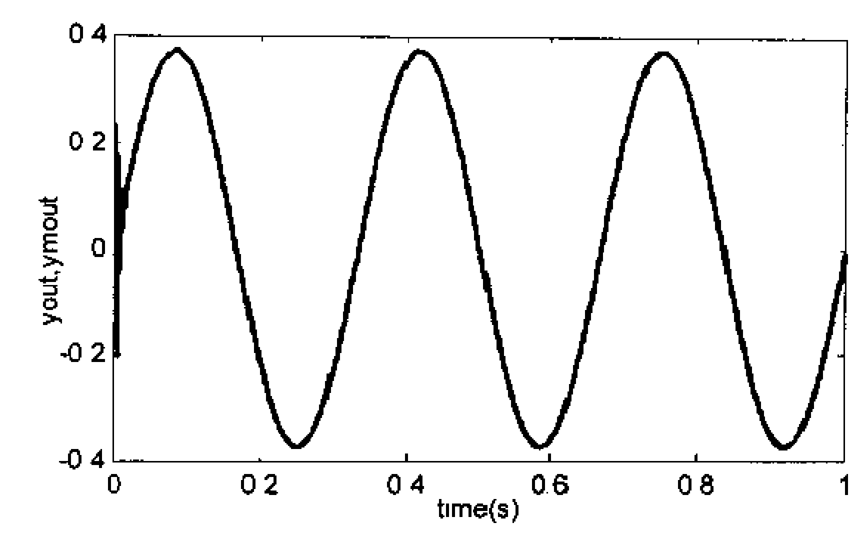


图 4-22　参考模型辨识结果

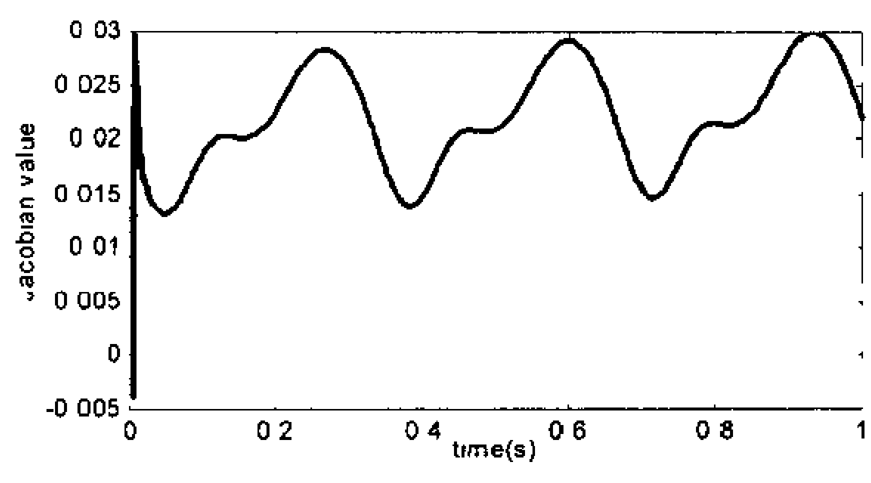


图 4-23　Jacobian 信息辨识结果

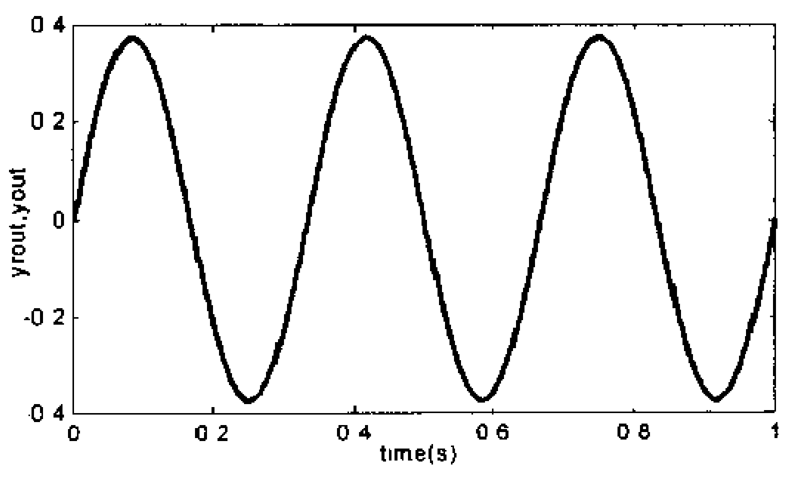


图 4-24　正弦位置跟踪

仿真程序：chap4_5.m。

```
%Single Neural Net PID Controller based on RBF Identification
clear all;close all;

Jp=0.0030;bp=0.067;
ts=0.001;

Gp=tf([1],[Jp,bp,0]);
Gpz=c2d(Gp,ts,'z');
[num,den]=tfdata(Gpz,'v');

h=zeros(6,1);
%w=rands(6,1);
w=[ 0.5646;
    0.3937;
    0.5556;
    0.3981;
    0.4495;
    0.2565];
w_1=w;w_2=w;w_3=w;

xite=0.40;
alfa=0.05;
belte=0.01;

x=[0,0,0]';

%c=0.1*ones(3,6);
%b=0.1*ones(6,1);
c=[ 3.1829    0.5211    7.1754   11.6631    3.6992   10.9150;
   -3.8909    2.3999    5.1730    8.5871   11.3737    7.0179;
    4.2018    2.6742    5.1828    8.5238    1.8936   -6.1845];
b=[ 5.3074;
    1.4771;
   26.4114;
   22.1716;
   52.9082;
    5.6906];
c_1=c;c_2=c_1;c_3=c_2;
b_1=b;b_2=b_1;b_3=b_2;
```

```
xc=[0,0,0]';
xitec=0.60;

kp=80;
kd=5;
ki=50;

wc=[kp,kd,ki];
wc_1=wc;wc_2=wc;wc_3=wc;

error_1=0;error_2=0;
y_1=0;y_2=0;
u_1=0;u_2=0;

ei=0;
c_size=size(c);

for k=1:1:1000
   time(k)=k*ts;
   rin(k)=0.50*sin(3*2*pi*k*ts);

S=2;
if S==1
  yrout(k)=1.0*rin(k);
end
if S==2
    yrout(k)=0.2*y_1+0.6*rin(k); %Reference Model
end

%Linear model
yout(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;

for j=1:1:c_size(2),
   h(j)=exp(-norm(x-c_1(:,j))^2/(2*b_1(j)*b_1(j)));
end
   ymout(k)=w_1'*h;

id=abs(yout(k)-ymout(k));
if id>0.0001,
```

```
%------------------Adjusting RBF parameters-------------------%
   d_w=0*w;          % Defining matrix number of d_w equal to that of w
   for j=1:1:6
   d_w(j)=xite*(yout(k)-ymout(k))*h(j);
   end
   w=w_1+d_w+alfa*(w_1-w_2)+belte*(w_2-w_3);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   d_b=0*b;
   for j=1:1:6

d_b(j)=xite*(yout(k)-ymout(k))*w_1(j)*h(j)*(b_1(j)^-3)*norm(x-c_1(:,j))^2;
   end
   b=b_1+ d_b+alfa*(b_1-b_2)+belte*(b_2-b_3);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   for j=1:1:6
     for i=1:1:3

d_c(i,j)=xite*(yout(k)-ymout(k))*w_1(j)*h(j)*(x(i)-c_1(i,j))*(b_1(j)^-2);
     end
   end
   c=c_1+d_c+alfa*(c_1-c_2)+belte*(c_2-c_3);
end
%%%%%%%%%%% Calculating Jacobian %%%%%%%%%%%
dyu=0;
for j=1:1:c_size(2)
   dyu=dyu+w(j)*h(j)*(-x(1)+c(1,j))/b(j)^2;
end
dyout(k)=dyu;
%%%%%%Parameters Return%%%%%%%
error(k)=yrout(k)-yout(k);
xc(1)=error(k);
xc(2)=(error(k)-error_1)/ts;
ei=ei+error(k)*ts;
xc(3)=ei;

u(k)=wc*xc;  %Control law
if u(k)>=10,
   u(k)=10;
end
if u(k)<=-10,
   u(k)=-10;
```

```
end

d_wc=0*wc;     % Defining matrix number of d_w equal to that of w
for j=1:1:3
    d_wc(j)=xitec*error(k)*xc(j)*dyout(k);
end
wc=wc_1+d_wc+alfa*(wc_1-wc_2)+belte*(wc_2-wc_3);

error_2=error_1;error_1=error(k);

u_2=u_1;u_1=u(k);
y_2=y_1;y_1=yout(k);

x(3)=y_2;
x(2)=y_1;
x(1)=u_1;

w_3=w_2;w_2=w_1;w_1=w;
c_3=c_2;c_2=c_1;c_1=c;
b_3=b_2;b_2=b_1;b_1=b;
wc_3=wc_2;wc_2=wc_1;wc_1=wc;
end
figure(1);
plot(time,yout,'r',time,ymout,'b');
xlabel('time(s)');ylabel('yout,ymout');
figure(2);
plot(time,dyout,'r');
xlabel('time(s)');ylabel('Jacobian value');
figure(3);
plot(time,yrout,'b',time,yout,'r');
xlabel('time(s)');ylabel('yrout,yout');
figure(4);
plot(time,yrout-yout,'r');
xlabel('time(s)');ylabel('control error');
```

4.5 基于 CMAC（神经网络）与 PID 的并行控制

4.5.1 CMAC 概述

小脑模型神经网络（Cerebellar Model Articulation Controller，CMAC）是一种表达复杂非线性函数的表格查询型自适应神经网络，该网络可通过学习算法改变表格的内容，具有信息

分类存储的能力。

CMAC把系统的输入状态作为一个指针，把相关信息分布式地存入一组存储单元。它本质上是一种用于映射复杂非线性函数的查表技术，具体做法是将输入空间分成许多分块，每个分块指定一个实际存储器位置；每个分块学习到的信息分布式地存储到相邻分块的位置上；存储单元数通常比所考虑问题的最大可能输入空间的分块数少得多，故实现的是多对一的映射，即多个分块映射到同样一个存储器地址上。

CMAC已被公认为是一类联想记忆神经网络的重要组成部分，它能够学习任意多维非线性映射。CMAC算法可有效地用于非线性函数逼近、动态建模、控制系统设计等。CMAC较其他神经网络的优越性体现在：

（1）它是基于局部学习的神经网络，它把信息存储在局部结构上，使每次修正的权很少，在保证函数非线性逼近性能的前提下，学习速度快，适合于实时控制；

（2）具有一定的泛化能力，即所谓相近输入产生相近输出，不同输入给出不同输出；

（3）连续（模拟）输入、输出能力；

（4）寻址编程方式，在利用串行计算机仿真时，它可使回响速度加快；

（5）作为非线性逼近器，它对学习数据出现的次序不敏感。

由于CMAC所具有的上述优越性能，使它比一般神经网络具有更好的非线性逼近能力，更适合于复杂动态环境下的非线性实时控制。

CMAC的基本思想在于：在输入空间中给出一个状态，从存储单元中找到对应于该状态的地址，将这些存储单元中的内容求和得到CMAC的输出；将此响应值与期望输出值进行比较，并根据学习算法修改这些已激活的存储单元的内容。

CMAC的结构如图4-25所示。

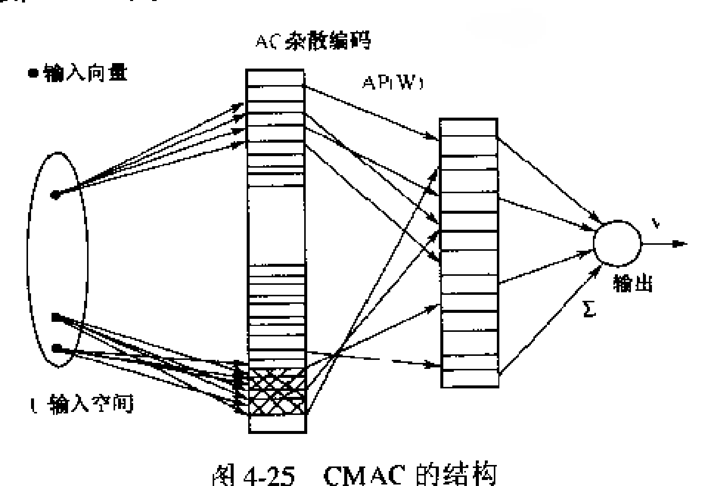


图4-25　CMAC的结构

4.5.2　一种典型CMAC算法及其仿真

CMAC网络由输入层、中间层和输出层组成。在输入层与中间层、中间层与输出层之间分别为由设计者预先确定的输入层非线性映射和输出层权值自适应性线性映射。

在输入层对n维输入空间进行划分。中间层由若干个基函数构成，对任意一个输入只有少数几个基函数的输出为非零值，称非零输出的基函数为作用基函数，作用基函数的个数为泛化参数c，它规定了网络内部影响网络输出的区域大小。

中间层基函数的个数用p表示，泛化参数c满足$c \ll p$。在中间层的基函数与输出层的

网络输出之间通过连接权进行连接。采用梯度下降法实现权值的调整。

CMAC 神经网络的设计主要包括输入空间的划分、输入层至输出层非线性映射的实现及输出层权值学习算法。

CMAC 是前馈网络，输入输出之间的非线性关系由以下两个基本映射实现。

1. 概念映射（U→AC）

概念映射是从输入空间 U 至概念存储器 AC 的映射。

设输入空间向量为 $\boldsymbol{u}_{\mathrm{p}}=[\boldsymbol{u}_{1\mathrm{p}},\boldsymbol{u}_{2\mathrm{p}},\cdots,\boldsymbol{u}_{n\mathrm{p}}]^{\mathrm{T}}$，量化编码为 $[u_{\mathrm{p}}]$，输入空间映射至 AC 中 c 个存储单元（c 为二进制非零单元的数目）。

采用下式表示映射后的向量：

$$\boldsymbol{R}_{\mathrm{p}}=S([u_{\mathrm{p}}])=[s_1(u_{\mathrm{p}}),s_2(u_{\mathrm{p}}),\cdots,s_c(u_{\mathrm{p}})]^{\mathrm{T}} \tag{4.50}$$

式中 $s_j([u_{\mathrm{p}}])=1$，$j=1,2,\cdots,c$。

映射原则：在输入空间邻近的两个点（一个点表示一输入的 n 维向量），在 AC 中有部分的重叠单元被激励。距离越近，重叠越多；距离越远，重叠越少。这种映射称为局部泛化，c 为泛化参数。

2. 实际映射（AC→AP）

实际映射是由概念存储器 AC 中的 c 个单元，用编码技术（如杂散编码）映射至实际存储器 AP 的 c 个单元，c 个单元中存放着相应权值。网络的输出为 AP 中 c 个单元的权值之和。

若只考虑单输出，则输出为：

$$y=\sum_{j=1}^{c}w_j s_j([u_{\mathrm{p}}]) \tag{4.51}$$

即：

$$y=\sum_{j=1}^{c}w_j \tag{4.52}$$

其中 $w_{\mathrm{p}}=[w_1 \quad w_2 \quad \cdots \quad w_c]^{\mathrm{T}}$。

CMAC 采用的学习算法如下：

采用 δ 学习规则调整权值，权值调整指标为：

$$E=\frac{1}{2c}e(t)^2 \tag{4.53}$$

式中，$e(t)=r(t)-y(t)$。

由梯度下降法，权值按下式调整：

$$\Delta w_j(t)=-\eta\frac{\partial E}{\partial w}=\eta\frac{(r(t)-y(t))}{c}\frac{\partial y}{\partial w}=\eta\frac{e(t)}{c} \tag{4.54}$$

$$w_j(t)=w_j(t-1)+\Delta w_j(t)+\alpha(w_j(t-1)-w_j(t-2)) \tag{4.55}$$

$$w_{\mathrm{p}}=[w_1 \quad w_2 \quad \cdots \quad w_c]^{\mathrm{T}} \tag{4.56}$$

式中，α 为惯性系数。

4.5.3 仿真程序及分析

仿真实例

采用 CMAC 网络辨识非线性对象：

$$y(k)=u(k-1)^3+y(k-1)/(1+y(k-1)^2)$$

取 $u(k)$作为网络的输入，采用线性化函数对输入状态进行量化，实现 CMAC 的概念映射：

$$s(k)=\text{round}\left[\left(u(k)-x_{\min}\right)\frac{M}{x_{\max}-x_{\min}}\right]$$

式中，$x_{\min}$ 和 $x_{\max}$ 为输入的最大、最小值，M 为 $x_{\max}$ 量化后所对应的最大值。

采用杂散编码技术中的除留余数法实现 CMAC 的实际映射。设杂凑表长为 m，以元素值 $s(k)+i$ 除以某数 N（$N<m$）后，所得余数加 1 作为杂凑地址，实现了实际映射，即：

$$\text{ad}(i)=(s(k)+i\ \text{MOD}\ N)+1$$

式中，$i=1,2,\cdots,c$。

在仿真中，取 $M=100$，$N=7$，取泛化参数 $c=7,\eta=1.5,\alpha=0.05$。仿真结果如图 4-26 所示。

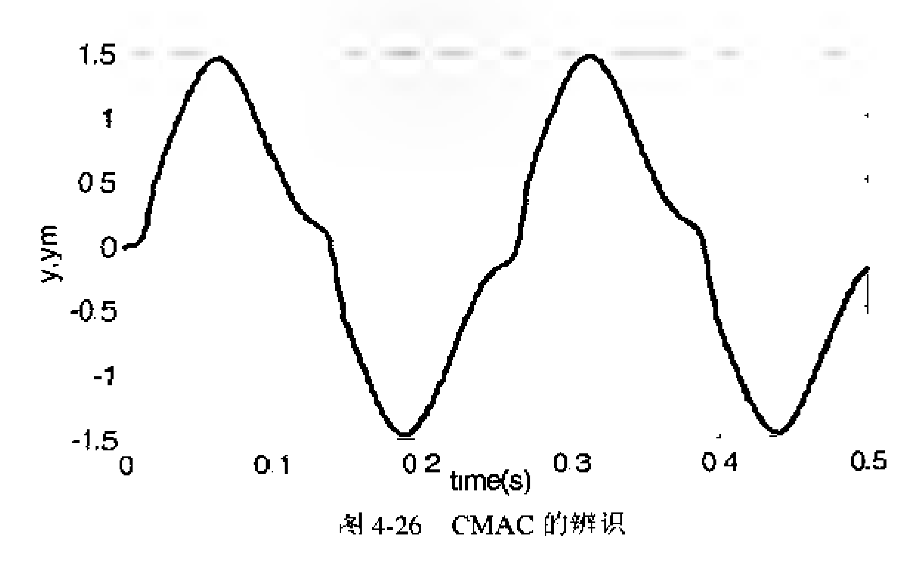


图 4-26 CMAC 的辨识

仿真程序：chap4_6.m

```
%CMAC Identification for nonlinear model
clear all;
close all;

xite=1.5;
alfa=0.05;

M=100;
N=7;
C=7;

w=zeros(N,1);
%w=rands(N,1);
```

```
w_1=w;
w_2=w;
d_w=w;

u_1=0;
y_1=0;
ts=0.001;

for k=1:1:500
time(k)=k*ts;

u(k)=sin(4*2*pi*k*ts);

xmin=-1.0;
xmax=1.0;

s(k)=round((u(k)-xmin)*M/(xmax-xmin));    %Quantity

sum=0;
for i=1:1:C
   ad(i)=mod(s(k)+i,N)+1;   %Table mapping and Hash transfer:Start address
   sum=sum+w(ad(i));
end

ym(k)=sum;

%Nonlinear model
y(k)=u_1^3+y_1/(1+y_1^2);

error(k)=y(k)-ym(k);
d_w=xite*error(k)/C;

for i=1:1:C
   ad(i)=mod(s(k)+i,N)+1;
   w(ad(i))=w_1(ad(i))+ d_w+alfa*(w_1(ad(i))-w_2(ad(i)));
end

%%%% Parameters Update %%%%
w_2=w_1;
w_1=w;
```

```
u_1 ,(K ;
y 1 y K ;
end
fig.re,1);
plot time,y,'b',time,ym, r');
xlabel 'time(s ',;ylabel('y,ym' ;
```

4.5.4 CMAC 与 PID 复合控制算法

CMAC 一开始就被应用于机器人控制中，目前有多种控制形式，如 CMAC 直接逆运动控制、CMAC 前馈控制、CMAC 反馈控制等，本书采用的是 CMAC 前馈控制。CMAC 与 PID 复合控制结构图如图 4-27 所示，该系统通过 CAMC 和 PID 的复合控制实现前馈反馈控制，其特点为：

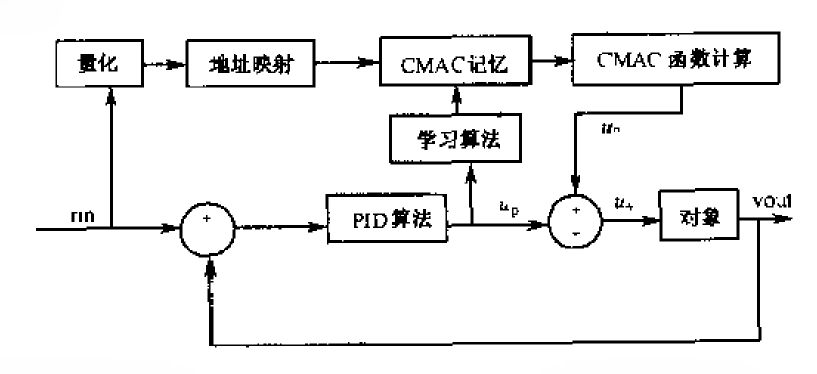


图 4-27 CMAC 与 PID 复合控制结构图

（1）小模型神经控制器实现前馈控制，实现被控对象的逆动态模型；

（2）常规控制器实现反馈控制，保证系统的稳定性，且抑制扰动。

CMAC 采用有导师的学习算法。每一控制周期结束时，计算出相应的 CMAC 输出 $u_n(k)$，并与总控制输入 $u(k)$ 相比较，修正权重，进入学习过程。学习的目的是使总控制输入与 CMAC 的输出之差最小。经过 CMAC 的学习，使系统的总控制输出由 CMAC 产生。而常规控制器采用传统的 PD 算法而不用 PID 控制算法，使 CMAC 的学习仅仅依赖于误差的当时测量值及变化值。

该系统的控制算法为：

$$u_n(k)=\sum_{i=1}^{c} w_i a_i \tag{4.57}$$

$$u(k)=u_n(k)+u_p(k) \tag{4.58}$$

式中，a_i 为二进制选择向量，c 为 CMAC 网络的泛化参数，$u_n(k)$ 为 CMAC 产生相应的输出，$u_p(k)$ 为常规控制器 PID 产生的输出。

在这里，CMAC 概念映射的方法为：输入空间 S 在区间$[S_{min}, S_{max}]$上分成 $N+2C$ 个量化间隔，即：

$$
\begin{aligned}
&v_1 = \cdots = v_c = S_{\min} \\
&v_j = v_{j-1} + \Delta v_j \qquad (j = c+1, \cdots, c+N) \\
&v_{N+c+1} = \cdots = v_{N+2c} = S_{\max}
\end{aligned}
\tag{4.59}
$$

CMAC 实际映射的方法为：

$$
a_j = \begin{cases} 1 & 若 S_j \in [v_j \quad v_{j+c}],\ j = c+1, \cdots, c+N \\ 0 & 其他 \end{cases}
\tag{4.60}
$$

取指令信号 rin(k) 作为 CMAC 的输入。每一控制周期结束时，CMAC 输出 $u_n(k)$ 与总控制输出 $u(k)$ 相比较，修正权重，进入学习过程。学习的目的是使总控制输入与 CMAC 的输出之差最小，即使系统的总控制输出主要由 CMAC 控制器产生。

CMAC 的调整指标为：

$$
E(k) = \frac{1}{2}(u_n(k) - u(k))^2 \cdot \frac{1}{c}
\tag{4.61}
$$

$$
\Delta w(k) = -\eta \frac{\partial E(k)}{\partial w} = \eta \frac{u(k) - u_n(k)}{c} a_i = \eta \frac{u_p(k)}{c} a_i
\tag{4.62}
$$

$$
w(k) = w(k-1) + \Delta w(k) + \alpha(w(k) - w(k-1))
\tag{4.63}
$$

式中，η 为网络学习速率，$\eta \in (0,1)$，α 为惯性量，$\alpha \in (0,1)$。

当系统开始运行时，置 $w = 0$，此时 $u_n = 0$，$u = u_p$，系统由常规控制器进行控制。通过 CMAC 的学习，使 PID 产生的输出控制量 $u_p(k)$ 逐渐为零，CMAC 产生的输出控制量 $u_n(k)$ 逐渐逼近控制器总输出 $u(k)$。

4.5.5 仿真程序及分析

仿真实例

被控对象采用二阶传递函数：

$$
G_p(s) = \frac{1770}{s^2 + 60s + 1770}
$$

CMAC 神经网络参数取 $N = 100$，$C = 5$，$\eta = 0.10$，$\alpha = 0.04$。PID 控制参数取 $k_p = 25, k_i = 0, k_d = 0.28$，采样时间为 1ms。取输入信号为方波信号，$M = 2$，跟踪结果如图 4-28 和图 4-29 所示，其中图 4-29 分别给出 CMAC 控制器、PD 控制器和总控制器的输出。

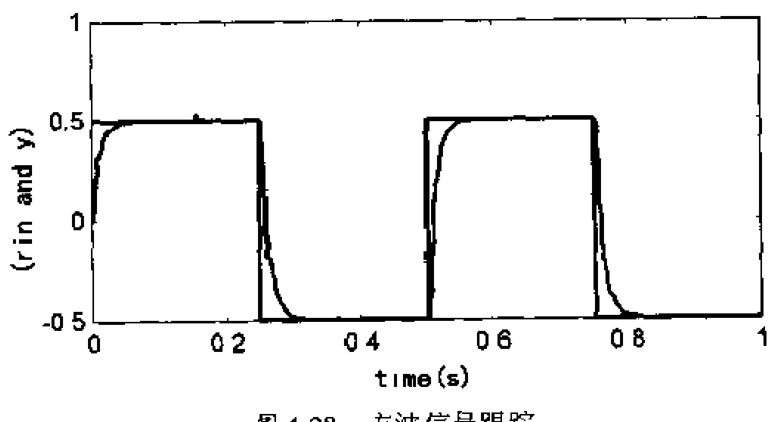


图 4-28　方波信号跟踪

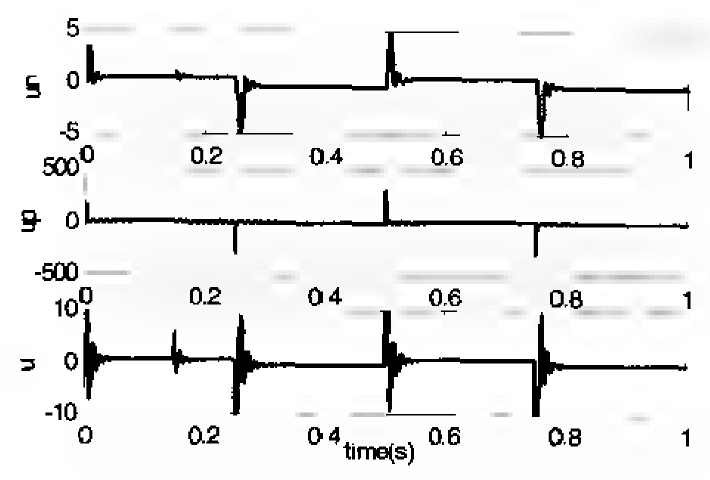


图 4-29　各个控制器的输出

仿真程序：chap4_7.m。

```
%CMAC and PID Concurrent Control
clear all;
close all;

ts=0.001;
sys=tf(1770,[1,60,1770]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

alfa=0.04;

N=100;C=5;

w=zeros(N+C,1);
w_1=w;w_2=w;d_w=w;

y_1=0;y_2=0;y_3=0;
u_1=0.0;u_2=0.0;u_3=0.0;

x=[0,0,0]';
error_1=0;

%Square Wave Signal
    A=0.50;
    Smin=-A;
    Smax=A;
    xite=0.10;
```

```
    kp=25;
    ki=0.0;
    kd=0.28;

%Coding Input value
dvi=(Smax-Smin)/(N-1);

for i=1:1:C                  %C number
   v(i)=Smin;
end
for i=C+1:1:C+N              %N number
   v(i)=v(i-1)+dvi;
end
for i=N+C+1:1:N+2*C          %C number
   v(i)=Smax;
end

for k=1:1:1000
   time(k)=k*ts;

rin(k)=A*sign(sin(2*2*pi*k*ts));       %Square Signal

for i=1:1:N+C
if rin(k)>=v(i)&rin(k)<=v(i+C)
   a(i)=1;
else
   a(i)=0;
end
end

yout(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;

error(k)=rin(k)-yout(k);

%CMAC Neural Network Controller
un(k)=a*w;

%PID Controller
up(k)=kp*x(1)+kd*x(2)+ki*x(3);
```

```
M 2·
if M  1      %Only Using PID Control
   u(k) up k ;
elseif M  2 %Total control output
     u(k -up k +un k ;
end

if k  150     %Disturbance
   u k) u k)+5.0;
end

if u(k > 1
     k  10;
end
if u k <  10
   u(k)  10);
end

%Update NN Weight
d_w a *xite*(u(k) un(k ) C;

w=w 1+ d_w+alfa* w_1 w 2);

%Parameters Update
w 3=w 2;w 2=w_1;w_1 w;
u 2 u 1,u_1 u(k ;
y 2 y 1;y_1 yout(k ;

x 1  error(k ;                 % Calculating P
x 2   error k) error 1  ts;   % Calculating D
x 3  x 3)+error(k *ts;         % Calculating I

error_2 error 1;error_1 error(k);
end
figure(1);
plot time,rin, k',time,yout,'k');
xlabel  time s)' ;ylabel('rin and y  );
figure 2);
subplot(311 ;
plot time,un,'k' ;
```

```
xlabel('time(s)');ylabel('rin');
subplot(3,1,2);
plot(time,up,'k');
xlabel('time(s)');ylabel('up');
subplot(3,1,3);
plot(time,u,'k');
xlabel('time(s)');ylabel('u');
figure(3);
plot(time,error,'k');
xlabel('time(s)');ylabel('error');
```

CMAC 控制算法虽然是由 PD 控制器的输出训练的，但并不是 PD 控制器的简单复制。加入 PD 控制器是为了评判 CMAC 控制器的性能，增强系统的稳定性，抑制扰动。PD 单独控制时，增益 k_p 的值在很大程度上决定着控制效果，而采用 PD+CMAC 控制时控制效果不依赖于 k_p 的值，k_p 的值只需在一个合理的范围内即可。

通过仿真结果可以看出，开始的时候主要是常规 PD 控制器起作用，经过对常规控制器的输出的不断学习，逐渐由小脑模型的输出起控制作用。小脑模型的加入使得控制效果比单独的 PID 控制效果要好很多，当方波（阶跃）输入时，大大减小了超调，加快控制响应速度，充分体现了小脑模型的特点，即输出误差小、实时性好、鲁棒性强等。

在加入干扰的情况下，会发现由于 CMAC 的加入使得干扰作用下的控制系统很快地恢复稳定状态。CMAC+PD 并行控制在一定程度上克服了常规控制器所不能避免的一些弊端，使控制效果得到提高。

4.6 CMAC 与 PID 并行控制的 Simulink 仿真

4.6.1 Simulink 仿真方法

采用 Simulink 仿真实现 CAMC 和 PID 的并行控制。通过 Simulink 仿真，可以使得控制结构看来更加清晰。

采用调用 *M* 函数的形式来编写 CMAC 神经网络控制器和 PID 控制器，指令信号类型是在 *M* 函数中实现的。通过选择 *S* 值不同的输入信号来实现阶跃、方波和正弦信号的跟踪。

4.6.2 仿真程序及分析

仿真实例

取采样时间为 1ms，被控对象采用二阶传递函数 $G_p(s)=1770/(s^2+60s+1770)$ 的离散方式为：

$$G_p(z)=\frac{0.0008674z+0.0008503}{z^2-1.94z+0.9418}$$

[illegible] Simulink [illegible] CMAC 和 PID [illegible]两个 M 函数来实现，即 chap4_8m1.m 和 chap4_8m2.m

在 PID 控制 M 函数中采用了 persistent 命令实现[illegible]，采用 global 命令实现参数的共享，[illegible] Clock 实现参数的初始化，[illegible]的类型，其中 $S=1$ 时，输入指令[illegible]为阶跃信号，[illegible]。

在 CMAC 控制 M 函数中利用时钟 Clock 功能实现[illegible]，CMAC 神经网络参数取 $N=300$，$C=5$，$\eta=0.10$，$\alpha=0.04$

Simulink 仿真时间取 1s，仿真[illegible]

(1) 阶跃响应：在 Simulink 程序中[illegible] Step 输入，在 M 函数中取 $S=1$，PID 参数取 $k_p=0.40$，$k_i=0$，$k_d=0.28$，[illegible] 4-30 [illegible]

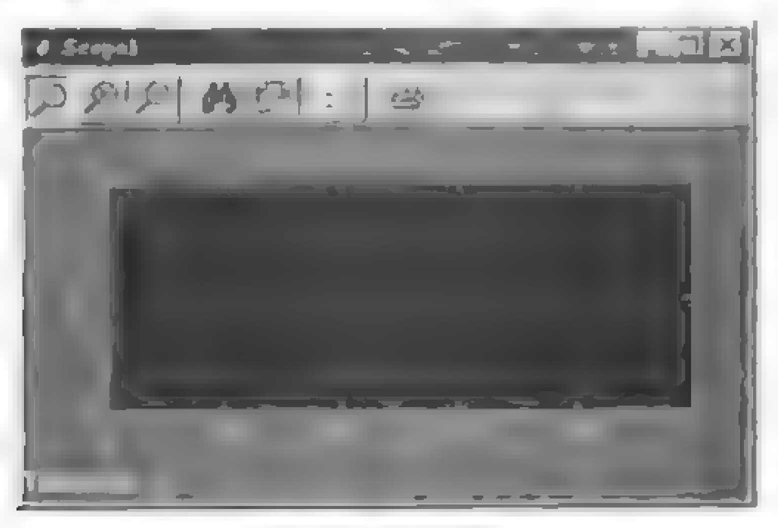


图 4-30 [illegible]

(2) 方波响应：在 Simulink [illegible]，[illegible] 0.50，[illegible]为 4Hz，在 M 函数中取 $S=3$，PID 参数取 $k_p=0.01$，$k_i=0$，$k_d=0.28$，方波响应[illegible]图 4-31 所示。

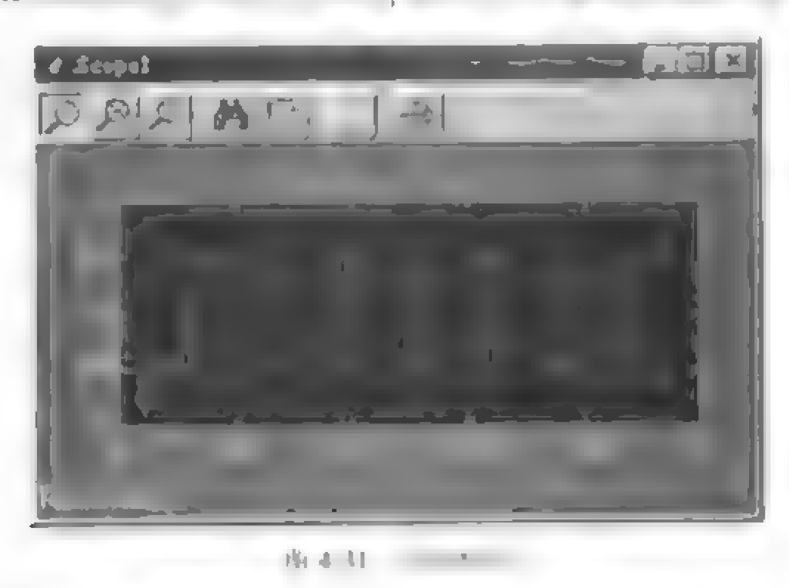


图 4-31 [illegible]

仿真程序 1：Simulink 程序 chap4_8.mdl，如图 4-32 所示

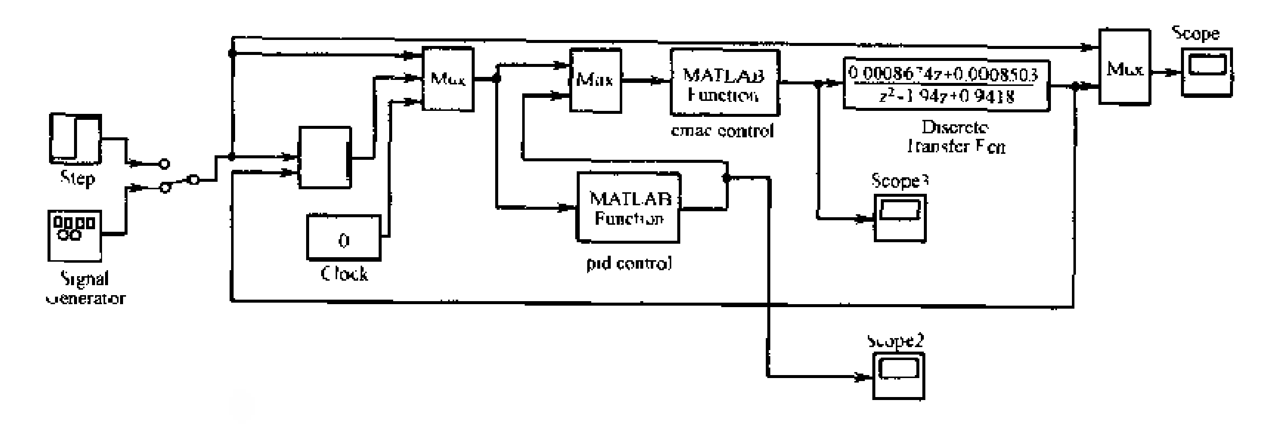


图4.32　Simulink控制仿真结构图

仿真程序2：用于PID控制的*M*函数chap4_8m1.m。

```
function [u]=chap4_8m1(u1,u2,u3)
global s
persistent errori error_1

ts=0.001;
if u3==0
   errori=0;
   error_1=0;
end

s=2;             %Selecting Signal Type
if s==1          %Step Signal
   kp=0.4;
   ki=0.0;
   kd=0.28;
elseif s==2      %Square Wave Signal
   kp=0;
   ki=0;
   kd=0.28;
end
error=u2;
errord=(error-error_1)/ts;
errori=errori+error*ts;

u=kp*error+kd*errord+ki*errori;
error_1=error;
```

仿真程序3：用于CMAC控制的*M*函数chap4_8m2.m。

```
function [u]=chap4_8m2(u1,u2,u3,u4)
global s
persistent w x1 x2 x3 w_1 w_2 w_3
```

```
N 300;
C 5;

if ... -0
   w zeros N+C,1 ;
   w 1-w;
   w_2 w;
   1 w w;
end

alfa 0 04;

if s  1       %Step Signal
   Smin 0;
   Smax 1;
elseif s  2   %Square Wave Signal
   Smin= 0.5;
   Smax-0.5;
end

xite-0.1;

dvi  Smax-Smin  (N 1 ;

for i 1:1:C                 %C size
    v(i) Smin;
end
for i C+1:1:C+N             %N size
    v i)-v i-1)+dvi;
end
for i-N+C+1.1:N+2*C          %C size
    v(i) Smax;
end

rin  i1;
for i-1:1:N+C
if rin> v(i)&rin<-v(i+C)
   a(i) 1;
else
   a i)-0;
end
end
```

```
errork ..2;

.n a*w 1;
.p .4;
. .p+.n;                %Total con ro o. p.t

1: .. ^
   . . ;
end
1f .. .0
   . .0;
end

d_w . *x.te*.p ..

w w_1+d_w+alfa* w 1 w 2 ,
w_3 w_2,
w 2 w ..,
w 1 w;
```

4.7 基于 Hopfield 网络的 PID 模型参考自适应控制

4.7.1 系统描述

典型的伺服系统速度环框图如图 4-33 所示。

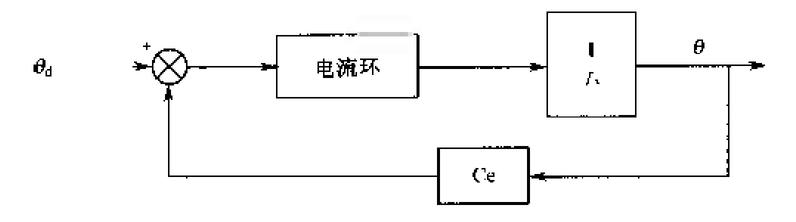


图 4-33 伺服系统速度环框图

速度环动态方程为：

$$I\times\frac{1}{Js}=\dot{\theta}$$

即：

$$\dot{v}=\frac{1}{J}\cdot I \qquad (4.64)$$

式中，J 为转动惯量，I 为电流控制输入，速度指令为 $v_d=\dot{\theta}_d$，实际速度为 $v=\dot{\theta}$。

理想参考模型定义为：

$$\dot{v}_{m} = -\frac{1}{T}v_{m} + \frac{1}{T}v_{d} \tag{4.65}$$

式中，v_m 为参考模型的速度。

参考文献[12]，将电流环控制器设计为“P 控制+前馈控制”的形式：

$$I = k_{p}(v_{d} - v) + k_{f}v_{d} \tag{4.66}$$

将式（4.66）带入式（4.64），并整理，得：

$$\dot{v} = \frac{k_{p}}{J}(v_{d} - v) + \frac{k_{f}}{J}v_{d} = -\frac{k_{p}}{J}v + \left(\frac{k_{p}}{J} + \frac{k_{f}}{J}\right)v_{d}$$

令：

$$K = \frac{1}{J}, \quad F = k_{p}, \quad G = k_{p} + k_{f} \tag{4.67}$$

得：

$$\dot{v} = -KFv + KGv_{d} \tag{4.68}$$

式中，F 和 G 为待定的控制器参数，采用 Hopfield 网络的输出加以整定。

4.7.2 基于 Hopfield 网络的控制器优化

Hopfield 网络结构如图 4-34 所示。

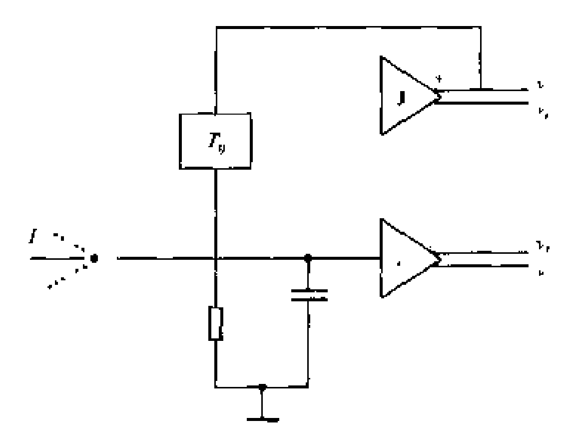


图 4-34 Hopfield 网络结构

Hopfield 网络的能量函数取：

$$E = \frac{1}{2}(\dot{v}_{m} - \dot{v})^{2} \tag{4.69}$$

将式（4.68）带入式（4.69）展开得：

$$E = \frac{1}{2}\left(\dot{v}_{m}^{2} + K^{2}F^{2}v^{2} + K^{2}G^{2}v_{d}^{2}\right) + \frac{1}{2}\left(2KF\dot{v}_{m}v - 2KGv_{d}\dot{v}_{m} - 2K^{2}FGvv_{d}\right)$$

取 Hopfield 网络输出神经元数为 2，假设输入电阻无穷大，此时 Hopfield 网络的标准能量函数为：

$$E_{N} = -\frac{1}{2}\sum_{i=1}^{2}\sum_{j=1}^{2}T_{ij}v_{i}v_{j} - \sum_{i=1}^{2}v_{i}I_{i} \tag{4.70}$$

将 E_N 展开得：

$$E_{\mathrm{N}} = -\frac{1}{2}\left(T_{11}v_1^2 + T_{12}v_1 v_2 + T_{21}v_2 v_1 + T_{22}v_2^2\right) - v_1 I_1 - v_2 I_2$$

令 $v_1 = F, v_2 = G$，得：

$$E_{\mathrm{N}} = -\frac{1}{2}\left(T_{11}F^2 + T_{12}FG + T_{21}FG + T_{22}G^2\right) - FI_1 - GI_2$$

当 Hopfield 网络处于平衡状态时，能量函数最小，$T_{12} = T_{21}$，此时

$$\frac{\partial E_{\mathrm{N}}}{\partial F} = \frac{\partial E}{\partial F} = 0$$

$$\frac{\partial E_{\mathrm{N}}}{\partial G} = \frac{\partial E}{\partial G} = 0$$

由 $\frac{\partial E_{\mathrm{N}}}{\partial F} = \frac{\partial E}{\partial F} = 0$，得：

$$\frac{\partial E_{\mathrm{N}}}{\partial F} = \frac{1}{2}\left(-2T_{11}F - T_{12}G - T_{21}G\right) - I_1 = \frac{1}{2}\left(-2T_{11}F - 2T_{12}G\right) - I_1 = 0$$

$$\frac{\partial E}{\partial F} = \frac{1}{2}\left(2K^2Fv^2 + 2Kv_{\mathrm{m}}v - 2K^2Gvv_{\mathrm{d}}\right) = 0$$

由上面两式得：

$$T_{11} = -K^2v^2，\quad T_{12} = T_{21} = K^2vv_{\mathrm{d}}，\quad I_1 = -K\dot{v}_{\mathrm{m}}v$$

由 $\frac{\partial E_{\mathrm{N}}}{\partial G} = \frac{\partial E}{\partial G} = 0$，得：

$$\frac{\partial E_{\mathrm{N}}}{\partial G} = \frac{1}{2}\left(-T_{12}F - T_{21}F - 2T_{22}G\right) - I_2 = \frac{1}{2}\left(-2T_{12}F - 2T_{22}G\right) - I_2 = 0$$

$$\frac{\partial E}{\partial G} = \frac{1}{2}\left(2k^2Gv_{\mathrm{d}}^2 - 2kv_{\mathrm{d}}\dot{v}_{\mathrm{m}} - 2k^2Fvv_{\mathrm{d}}\right) = 0$$

由上面两式得：

$$I_2 = kv_{\mathrm{d}}\dot{v}_{\mathrm{m}}，\quad T_{22} = -k^2v_{\mathrm{d}}^2$$

通过上面推导得到连接权矩阵 $\boldsymbol{T}$ 和外部输入 $\boldsymbol{I}$ 如下：

$$\boldsymbol{T} = \begin{bmatrix} -k^2v^2 & k^2vv_{\mathrm{d}} \\ k^2vv_{\mathrm{d}} & -k^2v_{\mathrm{d}}^2 \end{bmatrix}，\quad \boldsymbol{I} = \begin{bmatrix} -kv_{\mathrm{m}}v & kv_{\mathrm{d}}v_{\mathrm{m}} \end{bmatrix} \tag{4.71}$$

标准 Hopfield 网络的动态方程为：

$$C_i\frac{\mathrm{d}u_i}{\mathrm{d}t} = \sum_{j=1}^{n}T_{ij}v_j + I_i \tag{4.72}$$

$$v_i = g(u_i)$$

取 $C_i = 1.0$，将所求的 $\boldsymbol{T}$ 和 $\boldsymbol{I}$ 带入上式得：

$$\frac{\mathrm{d}u_1}{\mathrm{d}t} = T_{11}v_1 + T_{12}v_2 + I_1 = -k^2v^2g(u_1) + k^2vv_{\mathrm{d}}g(u_2) - k\dot{v}_{\mathrm{m}}v$$

$$\frac{\mathrm{d}u_2}{\mathrm{d}t} = T_{21}v_1 + T_{22}v_2 + I_2 = k^2vv_{\mathrm{d}}g(u_1) - k^2v_{\mathrm{d}}^2g(u_2) + k\dot{v}_{\mathrm{m}}v_{\mathrm{d}}$$

取神经元输出的非线性特性为：

$$g(u_i) = \frac{2S_i}{1 + \mathrm{e}^{-\lambda_i u}} - S_i \qquad i = 1,2 \tag{4.73}$$

网络实际输出为：

$$F = g(u_1)$$

$$G = g(u_2)$$

由于 $1+e^{-\lambda_1 u_1} = \dfrac{2S_1}{F+S_1}$，$1+e^{-\lambda_2 u_2} = \dfrac{2S_2}{G+S_2}$，则有：

$$\frac{dF}{du_1} = \frac{-2S_1 e^{-\lambda_1 u_1}(-\lambda_1)}{\left(1+e^{-\lambda_1 u_1}\right)^2} = 2\lambda_1 S_1 \frac{S_1 - F}{F+S_1}\frac{(F+S_1)^2}{(2S_1)^2} = \frac{\lambda_1\left(S_1^2 - F^2\right)}{2S_1}$$

$$\frac{dF}{dt} = \frac{dF}{du_1}\frac{du_1}{dt} = \frac{\lambda_1\left(S_1^2 - F^2\right)}{2S_1}\left(-K^2 v^2 F + K^2 v v_d G - K\dot{v}_m v\right) \tag{4.74}$$

同理可得：

$$\frac{dG}{du_2} = \frac{\lambda_2\left(S_2^2 - G^2\right)}{2S_2}$$

$$\frac{dG}{dt} = \frac{dG}{du_2}\frac{du_2}{dt} = \frac{\lambda_2\left(S_2^2 - G^2\right)}{2S_2}\left(k^2 v v_d F - k^2 v_d^2 G + k\dot{v}_m v_d\right) \tag{4.75}$$

求解微分方程式（4.74）和式（4.75），可得到优化后的 F、G，从而实现 k_p 和 k_f 的整定。

4.7.3 仿真程序及分析

仿真实例

设被控制对象为：

$$I \times \frac{1}{Js} = \dot{\theta}$$

式中，$J = \dfrac{1}{60}$，则 $K=1/J=60$。

仿真结果如图 4-35～图 4-38 所示。其中图 4-35 为速度跟踪结果，图 4-36 为控制信号的输入，图 4-37 和图 4-38 为控制器参数 k_p 和 k_f 的变化过程。

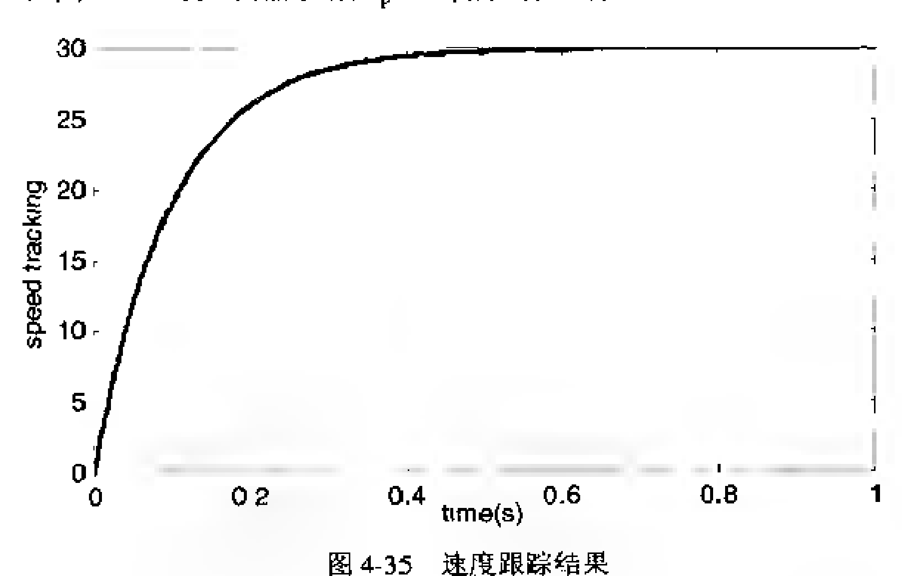


图 4-35 速度跟踪结果

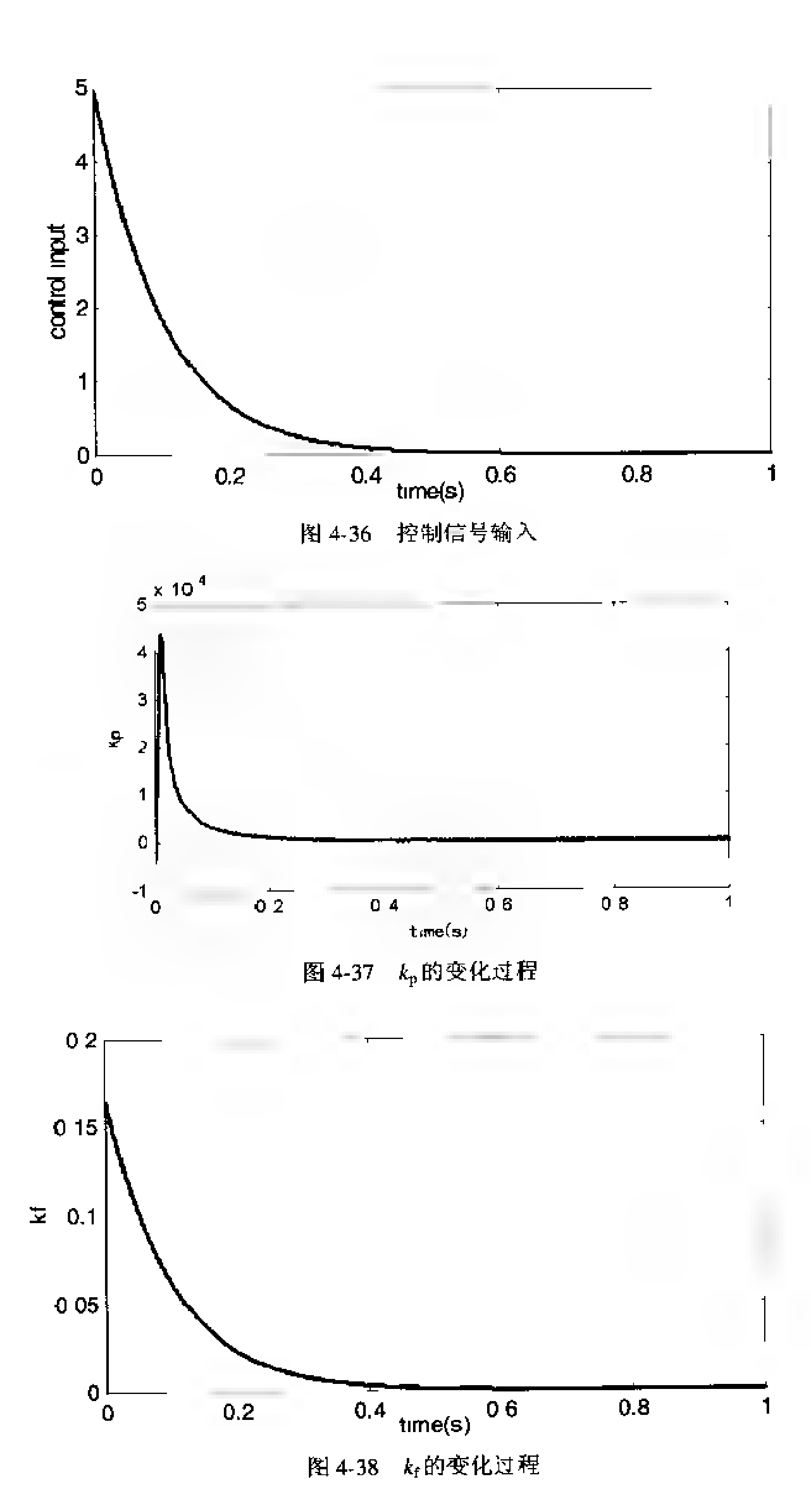


图 4-36　控制信号输入

图 4-37　k_p的变化过程

图 4-38　k_f的变化过程

仿真主程序：chap4_9.m。

```
%PID control based on Hopfield
```

```
clear all;
close all;

ts=0.001;
TimeSet=[0:ts:1];

para=[];

[t,x]=ode45('chap4_9eq',TimeSet,[0 0 0 0],[],para);
vm=x(:,1);
v=x(:,2);
F=x(:,3);
G=x(:,4);

vd=30;
I=F.*v+G*vd;

kp=F;
kf=-G kp;

figure(1);
plot(t,vm,'r',t,v,'k',t,vd,'k');
xlabel('time(s)');ylabel('speed tracking');

figure(2);
plot(t,I,'r');
xlabel('time(s)');ylabel('control input');

figure(3);
plot(t,kp,'r');
xlabel('time(s)');ylabel('kp');

figure(4);
plot(t,kf,'r');
xlabel('time(s)');ylabel('kf');
```

子程序：chap4_9eq.m

```
function dx=DynamicModel(t,x,flag,para)
dx=zeros(4,1);

v=x(2);
F=x(3);
G=x(4);
```

```
vd-30;

nmn1 0.01;
nmn2 0.01;

S1-1;
S2 1;

T-100*0.001;   %100ms
K 60;

I- F*v+G*vd;

dx(1   1 T*x(1,+1/T*vd;
dvm dx(1);
dx(2  K*I;
dx 3, nmn1 (2*S1)*(S1^2 F^2)*(-K^2*v^2*F+K^2*vd*v*G-K*dvm*v);
dx(4  nmn2 (2*S2)*(S2^2 G^2)*  K^2*v*vd*F K^2*vd^2*G+K*dvm*vd);
```

4.8 基于模糊 RBF 网络整定的 PID 控制

4.8.1 模糊神经网络结构

图 4-39 示出模糊 RBF 神经网络结构，该网络由输入层、模糊化层、模糊推理层及输出层构成。网络输出为 k_p，k_i，k_d。

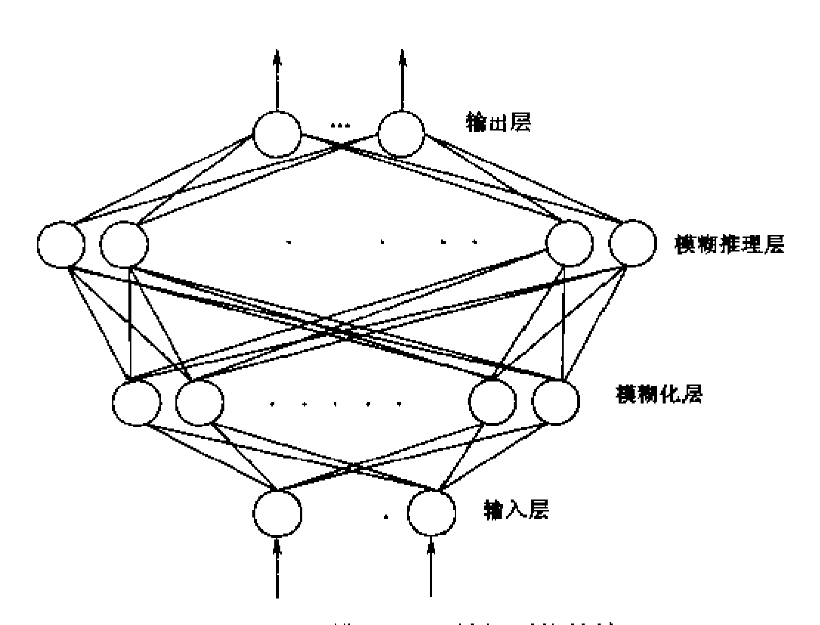


图 4-39　模糊 RBF 神经网络结构

在参考文献[13]中，将模糊 RBF 网络中信号传播和各层的功能表示如下：

第一层：输入层

输入层的各个节点直接与输入量的各个分量连接，将输入量传到下一层。对该层的每个节点 i 的输入输出表示为：

$$f(i)=X=[x_1,x_2,\cdots,x_n] \tag{4.76}$$

第二层：模糊化层

采用高斯型函数作为隶属函数，c_{ij} 和 b_{ij} 分别是第 i 个输入变量第 j 个模糊集合的隶属函数的均值和标准差

$$f_2(i,j)=\exp\left\{-\frac{(f_1(i)-c_{ij})^2}{(b_{ij})^2}\right\} \tag{4.77}$$

式中，$i=1,2,\cdots,n$；$j=1,2,\cdots,n$。

第三层：模糊推理层

模糊推理层通过与模糊化层的连接来完成模糊规则的匹配，各个节点之间实现模糊运算，即通过各个模糊节点的组合得到相应的点火强度。每个节点 j 的输出为该节点所有输入信号的乘积，即：

$$f_3(j)=\prod_{j=1}^{N} f_2(i,j) \tag{4.78}$$

式中，$N=\prod_{i=1}^{n} N_i$

第四层：输出层

输出层输出 f_4 为 k_p, k_i, k_d 整定结果，该层由三个节点构成，即：

$$f_4(i)=w\cdot f_3=\sum_{j=1}^{N} w(i,j)\cdot f_3(j) \tag{4.79}$$

式中，w_{ij} 组成输出节点与第三层各节点的连接权矩阵 $i=1,2,3$。

控制器为：

$$\Delta u(k)=f_4\cdot xc=k_p xc(1)+k_i xc(2)+k_d xc(3) \tag{4.80}$$

其中，

$$k_p=f_4(1),\quad k_i=f_4(2),\quad k_d=f_4(3)$$

$$xc(1)=e(k)$$

$$xc(2)=e(k)-e(k-1)$$

$$xc(3)=\Delta^2 e(k)=e(k)-2e(k-1)+e(k-2)$$

采用增量式 PID 控制算法：

$$u(k)=u(k-1)+\Delta u(k) \tag{4.81}$$

采用 Delta 学习规则来修正可调参数，定义目标函数为：

$$E=\frac{1}{2}(\mathrm{rin}(k)-\mathrm{yout}(k))^2 \tag{4.82}$$

式中，$\mathrm{rin}(k)$ 和 $\mathrm{yout}(k)$ 分别表示网络的实际输出和理想输出，每一个迭代步骤 k 的控制误差为 $\mathrm{rin}(k)-\mathrm{yout}(k)$。网络权值的学习算法如下：

$$\begin{aligned}\Delta w_j(k)&=-\eta\cdot\frac{\partial E}{\partial w_j}=\eta\cdot(\mathrm{rin}(k)-\mathrm{yout}(k))\cdot\frac{\partial \mathrm{yout}}{\partial \Delta u}\frac{\partial \Delta u}{\partial f_4}\frac{\partial f_4}{\partial w_j}\\&=\eta\cdot(\mathrm{rin}(k)-\mathrm{yout}(k))\cdot\frac{\partial \mathrm{yout}}{\partial \Delta u}xc(j)f_3(j)\end{aligned} \tag{4.83}$$

式中，w_j 为网络输出节点与上一层各节点的连接权，$j=1,2,\cdots,N$，η 为学习速率。

若考虑动量因子，则输出层的权值为：

$$w_j(k)=w_j(k-1)+\Delta w_j(k)+\alpha(w_j(k-1)-w_j(k-2)) \tag{4.84}$$

式中，k 为网络的迭代步骤，α 为学习动量因子。

4.8.2 仿真程序及分析

仿真实例

设被控制对象为：

$$y(k)=\frac{0.1y(k-1)+u(k-1)}{1+y(k-1)^2}$$

仿真中，网络的输入信号为两个，即指令信号和对象的实际输出，针对每个输入取 5 个模糊集进行模糊化，即 $n=2, N=5$，网络结构取 2-5-5-3 的形式，网络学习参数取 $\eta=0.20$，$\alpha=0.02$。网络的初始权值及隶属函数参数初值通过试验得到。仿真结果如图 4-40 和图 4-41 所示。

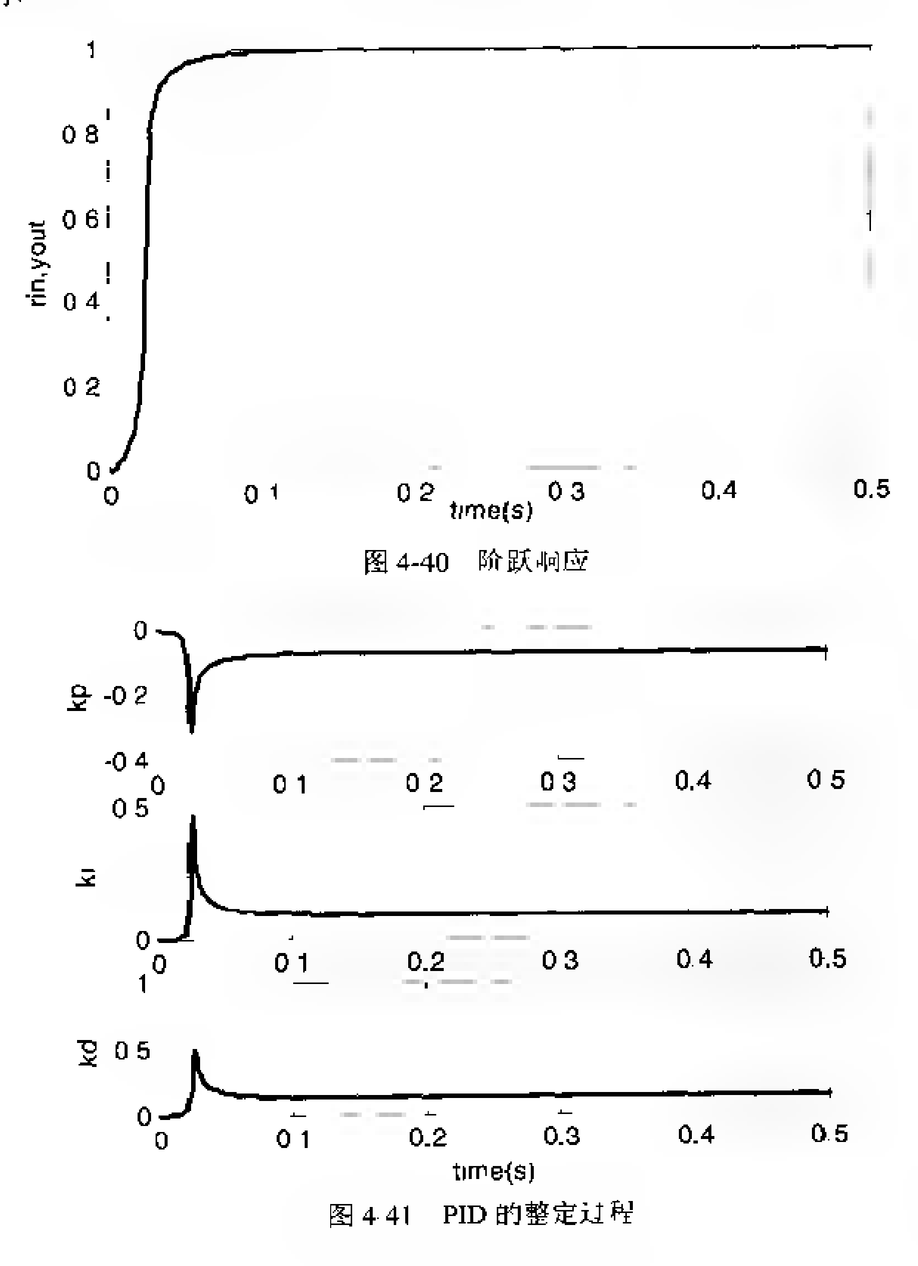


图 4-40 阶跃响应

图 4-41 PID 的整定过程

仿真程序：chap4_10.m。

```
%Adaptive PID control based on Fuzzy RBF Identification
clear all;
close all;

xite=0.20;
alfa=0.02;

c0=0.65*ones(2,5);
b0=1.0*ones(5,1);
w0=[0.5612   0.2176   0.5049
    0.55??   0.5065   0.7017
    0.7906   0.8590   0.3218
    0.3420   0.1411   0.5261
    0.4778   0.9020   0.45??
    0.2232   0.8098   0.4829
    0.?947   0.9565   0.0610
    0.5?36   0.3156   0.7150
    0.3114   0.4539   0.6068
    0.?907   0.0?2?   0.0480
    0.4553   0.3746   0.3?19
    0.1536   0.2741   0.0844
    0.?98    0.1980   0.1230
    0.6349   0.0287   0.8504
    0.6509   0.9090   0.4512
    0.1935   0.9827   0.4087
    0.?032   0.3606   0.2251
    0.6951   0.4871   0.?781
    0.3202   0.?2?2   0.4289
    0.9614   0.1060   0.?970
    0.5366   0.1631   0.78?7
    0.0842   0.1763   0.4761
    0.9129   0.428?   0.5240
    0.8009   0.7944   0.4420
    0.9138   0.2397   0.6904];
%w0=rands(25,3);

c=c0;c_1=c0;c_2=c0;
b=b0;b_1=b0;b_2=b0;
w4=w0;w4_1=w0;w4_2=w0;

x=[0,0]';
```

```
xc=[0,0,0]';

du_1=0;u_1=0;y_1=0;
error_1=0;error_2=0;error=0;

ts=0.001;
for k=1:1:50
   time(k)=k*ts;
   rin(k)=1;
   yout(k)=(0.[illegible]*y_1+u_1)/(1+y_1^2);   %Nonlinear plant

   x(1)=rin(k);
   x(2)=yout(k);

   f1=x;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:1:2                        % Layer2:fuzzation
   for j=1:1:5
      net2(i,j)=-(f1(i)-c_1(i,j))^2/b_1(j)^2;
   end
end
for i=1:1:2
   for j=1:1:5
       f2(i,j)=exp(net2(i,j));
   end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for j=1:1:5                       % Layer3:fuzzy inference(49 rules)
    m1(j)=f2(1,j);
     m2(j)=f2(2,j);
end

for i=1:1:5
for j=1:1:5
    ff3(i,j)=m2(i)*m1(j);
end
end
f3=[ff3(1,:),ff3(2,:),ff3(3,:),ff3(4,:),ff3(5,:)];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

f4=w4'*f3';                     % Layer4:output
```

```
kp κ  f4 1 ·
ki κ) f4 ₋ι;
κd k  f4 3 ;

%      -  Calcılate error ıd between yout and ymout      --   %
error(k -rın κ  yout k);

dı κ  κp k)*xc 1)+κd(k)*xc(2)+kı(k *xc(3 ;
.(k -u_1+du k ;

dy.(k  sıgn  yo.t k)-y 1)/ du(k) du_1+0.0001 );

d_w4 .0*w4_1;
for ı 1:1:25
  for  -1.1·5
    d w4 ı,ı  xıte*error(k)*dy.(k)*xc(j,*f3(i,;
  end
end
w4-w4 .+ d_w4+alfa*(w4_1 w4 2);

%Return of parameters
    d. 1-d. κ ;
  . 1 .(K ;
  y_1-yoıt(κ ;

  w4_2 w4 1;w4 1 w4;

  xc 1)-error(k,-error_1;                %Calc.latıng P
  xc(2 -error(k ,                        %Calculatıng I
  xc 3  error k -2*eııor_1+error_2;      %Calculatıng D

  error 2 error_1;
  error 1 error(k ;
end
fıgure(1 ;
plot(tıme,rın, b',tıme,yout,'r');
xlabel('tıme s ' ;ylabel('rın,yout',;
fıg.re 2);
s.bplot 311);
plot(tıme,kp,'r');
xlabel  tıme(s ',;ylabel 'kp');
subplot.312);
```

```
plot(time,ki,'r');
xlabel('time(s)');ylabel('ki');
subplot(313);
plot(time,kd,'r');
xlabel('time(s)');ylabel('kd');
```

第 5 章　基于遗传算法整定的 PID 控制

5.1　遗传算法的基本原理

遗传算法简称 GA（Genetic Algorithms）是 1962 年由美国 Michigan 大学的 Holland 教授提出的模拟自然界遗传机制和生物进化论而成的一种并行随机搜索最优化方法。它将"优胜劣汰，适者生存"的生物进化原理引入优化参数形成的编码串联群体中，按所选择的适配值函数并通过遗传中的复制、交叉及变异对个体进行筛选，使适配值高的个体被保留下来，组成新的群体，新的群体既继承了上一代的信息，又优于上一代。这样周而复始，群体中个体适应度不断提高，直到满足一定的条件。其算法简单，可并行处理，能得到全局最优解。

遗传算法的主要特点：

（1）遗传算法是对参数的编码进行操作，而非对参数本身；

（2）遗传算法是从许多点开始并行操作，而非局限于一点；

（3）遗传算法通过目标函数来计算适配值，而不需要其他推导，从而对问题的依赖性较小；

（4）遗传算法的寻优规则是由概率决定的，而非确定性的；

（5）遗传算法在解空间进行高效启发式搜索，而非盲目地穷举或完全随机搜索；

（6）遗传算法对于待寻优的函数基本无限制，它既不要求函数连续，也不要求函数可微，既可以是数学解析式所表示的显函数，又可以是映射矩阵甚至是神经网络的隐函数，因而应用范围较广；

（7）遗传算法具有并行计算的特点，因而可通过大规模并行计算来提高计算速度；

（8）遗传算法更适合大规模复杂问题的优化；

（9）遗传算法计算简单、功能强。

遗传算法的基本操作如下：

（1）复制（Reproduction Operator）。复制是从一个旧种群中选择生命力强的个体位串产生新种群的过程。根据位串的适配值复制，也就是指具有高适配值的位串更有可能在下一代中产生一个或多个子孙。它模仿了自然现象，应用了达尔文的适者生存理论。复制操作可以通过随机方法来实现。若用计算机程序来实现，可考虑首先产生 0～1 之间均匀分布的随机数，若某串的复制概率为 40%，则当产生的随机数在 0.40～1.0 之间时，该串被复制，否则被淘汰。此外，还可以通过计算方法实现，其中较典型的几种方法为适应度比例法、期望值法、排位次法等。适应度比例法较常用。

（2）交叉（Crossover Operator）。复制操作能从旧种群中选择出优秀者，但不能创造新的染色体，而交叉模拟了生物进化过程中的繁殖现象，通过两个染色体的交换组合，产生新的优良品种。它的过程为：在匹配池中任选两个染色体，随机选择一点或多点交换点位置；交换双亲染色体交换点右边的部分，即可得到两个新的染色体数字串。交换体现了自然界中信息交换的思想。交叉有一点交叉、多点交叉，还有一致交叉、顺序交叉和周期交叉。一点

交叉是最基本的方法，应用较广。它是指染色体切断点有一处，例：

A:101100 1110 →101100 0101

B:001010 0101 →001010 1110

（3）变异（Mutation Operator）。变异运算用来模拟生物在自然的遗传环境中由于各种偶然因素引起的基因突变，它以很小的概率随机地改变遗传基因（表示染色体的符号串的某一位）的值。在染色体以二进制编码的系统中，它随机地将染色体的某一个基因由 1 变为 0，或由 0 变为 1。若只有选择和交叉，而没有变异，则无法在初始基因组合以外的空间进行搜索，使进化过程在早期就陷入局部解而进入终止过程，从而影响解的质量。为了在尽可能大的空间中获得质量较高的优化解，必须采用变异操作

5.2 遗传算法的优化设计

5.2.1 遗传算法的构成要素

（1）染色体编码方法：基本遗传算法使用固定长度的二进制符号来表示群体中的个体，其等位基因由二值符号集{0,1}所组成。初始个体的基因值可用均匀分布的随机值来生成，如，$x = 100111001000101101$ 就可表示一个个体，该个体的染色体长度 $n = 18$ 。

（2）个体适应度评价：基本遗传算法与个体适应度成正比的概率决定当前群体中每个个体遗传到下一代群体中的概率多少，为正确计算这个概率，要求所有个体的适应度必须为正数或零，因此，必须先确定由目标函数值到个体适应度之间的转换规则。

（3）遗传算子：基本遗传算法使用下述三种遗传算子：

① 选择运算使用比例选择算子；

② 交叉运算使用单点交叉算子；

③ 变异运算使用基本位变异算子或均匀变异算子。

（4）基本遗传算法的运行参数：有下述 4 个运行参数需要提前设定：

M：群体大小，即群体中所含个体的数量，一般取为 20～100；

G：遗传算法的终止进化代数，一般取为 100～500；

P_c：交叉概率，一般取为 0.49～0.99；

P_m：变异概率，一般取为 0.0001～0.1。

5.2.2 遗传算法的应用步骤

对于一个需要进行优化的实际问题，一般可按下述步骤构造遗传算法：

第一步：确定决策变量及各种约束条件，即确定出个体的表现型 X 和问题的解空间。

第二步：建立优化模型，即确定出目标函数的类型及数学描述形式或量化方法。

第三步：确定表示可行解的染色体编码方法，即确定出个体的基因型 x 及遗传算法的搜索空间。

第四步：确定解码方法，即确定出由个体基因型 x 到个体表现型 X 的对应关系或转换方法。

第五步：确定个体适应度的量化评价方法，即确定出由目标函数值 $J(x)$ 到个体适应度函

数 $F(x)$ 的转换规则。

第六步：设计遗传算子，即确定选择运算、交叉运算、变异运算等遗传算子的具体操作方法。

第七步：确定遗传算法的有关运行参数，即 M,G,P_c,P_m 等参数。

5.3 遗传算法求函数极大值

利用遗传算法求 Rosenbrock 函数的极大值：

$$\begin{cases} f_2(x_1,x_2)=100(x_1^2-x_2)^2+(1-x_1)^2 \\ -2.048 \le x_i \le 2.048 \quad (i=1,2) \end{cases}$$

该函数有两个局部极大点，分别是 $f(2.048,\ -2.048)=3897.7342$ 和 $f(-2.048,\ -2.048)=3905.9262$，其中后者为全局最大点。

5.3.1 二进制编码遗传算法求函数极大值

求解该问题遗传算法的构造过程：

（1）确定决策变量和约束条件；

（2）建立优化模型；

（3）确定编码方法：用长度为 10 位的二进制编码串分别表示两个决策变量 x_1,x_2。10 位二进制编码串可以表示从 0～1023 之间的1024 个不同的数，故将 x_1,x_2 的定义域离散化为1023个均等的区域，包括两个端点在内共有1024 个不同的离散点。从离散点-2.048 到离散点2.048，依次让它们分别对应于从0000000000(0) ～ 1111111111(1023)之间的二进制编码。再将分别表示 x_1,x_2 的两个 10 位长的二进制编码串连接在一起，组成一个 20 位长的二进制编码串，就构成了这个函数优化问题的染色体编码方法。使用这种编码方法，解空间和遗传算法的搜索空间就具有一一对应的关系。例如：x:0000110111 1101110001就表示一个个体的基因型，其中前 10 位表示 x_1，后 10 位表示 x_2。

（4）确定解码方法：解码时需要将 20 位长的二进制编码串切断为两个 10 位长的二进制编码串，然后分别将它们转换为对应的十进制整数代码，分别记为 y_1 和 y_2。依据个体编码方法和对定义域的离散化方法可知，将代码 y_i 转换为变量 x_i 的解码公式为：

$$x_i = 4.096\times\frac{y_i}{1023}-2.048 \qquad (i=1,2) \tag{5.1}$$

例如，对个体 x:0000110111 1101110001，它由两个代码组成：

$$y_1=55,\ y_2=881$$

上述两个代码经过解码后，可得到两个实际的值：

$$x_1=-1.828,\ x_2=1.476$$

（5）确定个体评价方法：由于Rosenbrock 函数的值域总是非负的，并且优化目标是求函数的最大值，故可将个体的适应度直接取为对应的目标函数值，即：

$$F(x)=f(x_1,x_2) \tag{5.2}$$

选个体适应度的倒数作为目标函数：

$$J(x)=\frac{1}{F(x)} \tag{5.3}$$

（6）设计遗传算子：选择运算使用比例选择算子，交叉运算使用单点交叉算子，变异运算使用基本位变异算子。

（7）确定遗传算法的运行参数：群体大小 $M=80$，终止进化代数 $G=100$，交叉概率 $P_c=0.60$，变异概率 $P_m=0.10$。

上述 7 个步骤构成用于求 Rosenbrock 函数极大值优化计算的二进制编码遗传算法。

采用上述方法进行仿真，经过 100 步迭代，最佳样本为：

BestS = [0000000000000000000000]

即当 $x_1=-2.0480$，$x_2=-2.0480$ 时，Rosenbrock 函数具有极大值，极大值为 3905.9。

在遗传算法的优化过程中，目标函数 J 和适应度函数 F 的变化过程如图 5-1 和图 5-2 所示，由仿真结果可知，随着进化过程的进行，群体中适应度较低的一些个体被逐渐淘汰，而适应度较高的一些个体会越来越多，并且它们都集中在所求问题的最优点附近，从而搜索到问题的最优解。

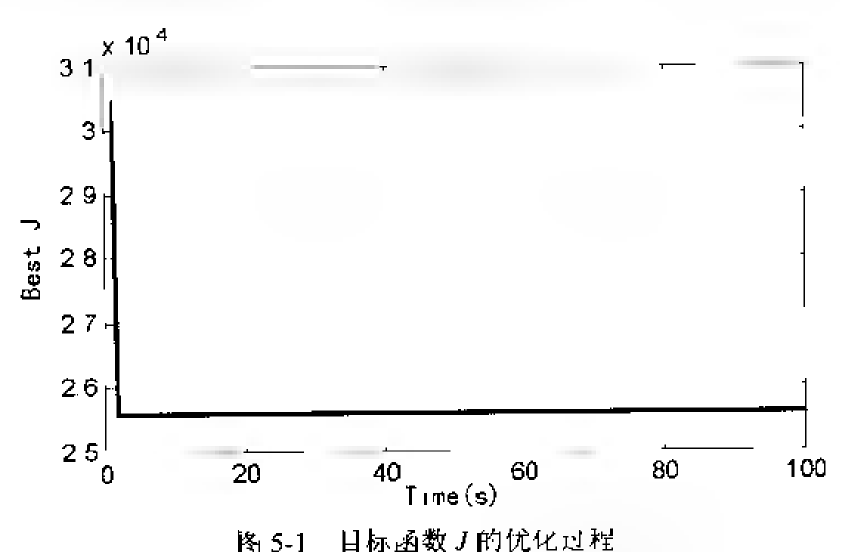


图 5-1　目标函数 J 的优化过程

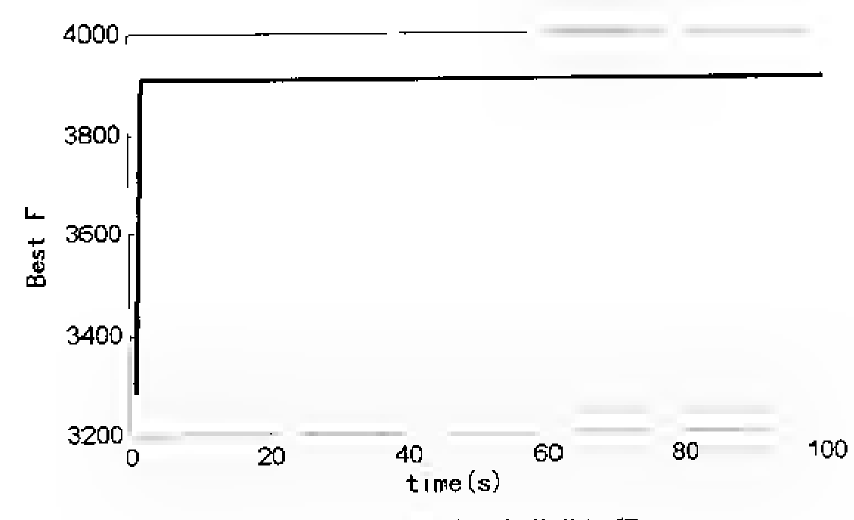


图 5-2　适应度函数 F 的优化过程

5.3.2 仿真程序

仿真程序：chap5_1.m。

```
%Generic Algorithm for function f(x1,x2) optimum
clear all;
close all;

%Parameters
Size=80;
G=100;
CodeL=10;

umax=2.048;
umin=-2.048;

E=round(rand(Size,2*CodeL));    %Initial Code

%Main Program
for k=1:1:G
time(k)=k;

for s=1:1:Size
m=E(s,:);
y1=0;y2=0;

%Uncoding
m1=m(1:1:CodeL);
for i=1:1:CodeL
   y1=y1+m1(i)*2^(i-1);
end
x1=(umax-umin)*y1/1023+umin;
m2=m(CodeL+1:1:2*CodeL);
for i=1:1:CodeL
   y2=y2+m2(i)*2^(i-1);
end
x2=(umax-umin)*y2/1023+umin;

F(s)=100*(x1^2-x2)^2+(1-x1)^2;
end

Ji=1./F;
```

```
%****** Step 1 : Evaluate BestJ ******
BestJ(k)=min(Ji);

fi=F;                          %Fitness Function
[Oderfi,Indexfi]=sort(fi);     %Arranging fi small to bigger
Bestfi=Oderfi(Size);           %Let Bestfi=max(fi)
BestS=E(Indexfi(Size),:);      %Let BestS=E(m), m is the Indexfi belong to
max(fi)
bfi(k)=Bestfi;

%****** Step 2 : Select and Reproduct Operation******
   fi_sum=sum(fi);
   fi_Size=(Oderfi/fi_sum)*Size;

   fi_S=floor(fi_Size);        %Selecting Bigger fi value

   kk=1;
   for i=1:1:Size
      for j=1:1:fi_S(i)        %Select and Reproduce
       TempE(kk,:)=E(Indexfi(i),:);
         kk=kk+1;              %kk is used to reproduce
      end
   end

%************ Step 3 : Crossover Operation ************
pc=0.60;
n=ceil(20*rand);
for i=1:2:(Size-1)
    temp=rand;
    if pc>temp                  %Crossover Condition
    for j=n:1:20
        TempE(i,j)=E(i+1,j);
        TempE(i+1,j)=E(i,j);
    end
    end
end
TempE(Size,:)=BestS;
E=TempE;

%************ Step 4: Mutation Operation **************
%pm=0.001;
%pm=0.001-[1:1:Size]*(0.001)/Size; %Bigger fi, smaller Pm
```

```
%pm=0.0;     %No mutation
pm=0.1;      %Big mutation

   for i=1:1:Size
      for j=1:1:2*CodeL
         temp=rand;
         if pm>temp                    %Mutation Condition
            if TempE(i,j)==0
               TempE(i,j)=1;
            else
               TempE(i,j)=0;
            end
        end
      end
   end

%Guarantee TempPop(30,:) is the code belong to the best individual(max(fi)),
TempE(Size,:)=BestS;
E=TempE;
end

Max_Value=Bestfi
BestS
x1
x2
figure(1);
plot(time,BestJ);
xlabel('Times');ylabel('Best J');
figure(2);
plot(time,bfi);
xlabel('times');ylabel('Best F');
```

5.3.3 实数编码遗传算法求函数极大值

求解该问题遗传算法的构造过程：

（1）确定决策变量和约束条件；

（2）建立优化模型；

（3）确定编码方法：用 2 个实数分别表示两个决策变量 x_1, x_2，分别将 x_1, x_2 的定义域离散化为从离散点 -2.048 到离散点 2.048 的 Size 个实数。

（4）确定个体评价方法：个体的适应度直接取为对应的目标函数值，即

$$F(x) = f(x_1, x_2) \tag{5.4}$$

取个体适应度的倒数作为目标函数

$$J(x)=\frac{1}{F(x)} \tag{5.5}$$

（5）设计遗传算子：选择运算使用比例选择算子，交叉运算使用单点交叉算子，变异运算使用基本位变异算子。

（6）确定遗传算法的运行参数：群体大小 $M=500$，终止进化代数 $G=200$，交叉概率 $P_c=0.90$，采用自适应变异概率 $P_m=0.10-[1:1:\mathrm{Size}]\times 0.01/\mathrm{Size}$，即变异概率与适应度有关，适应度越小，变异概率越大。

上述六个步骤构成了用于求 Rosenbrock 函数极大值的优化计算的实数编码遗传算法。

采用上述方法进行仿真，经过 200 步迭代，最佳样本为

$$\mathrm{Best}S=[-2.0438\quad -2.044]$$

即当 $x_1=-2.0438$，$x_2=-2.044$ 时，Rosenbrock 函数具有极大值，极大值为 3880.3。

遗传算法的优化过程中目标函数 J 和适应度函数 F 的变化过程如图 5-3 和图 5-4 所示，由仿真结果可知，采用实数编码的遗传算法搜索效率低于二进制遗传算法。

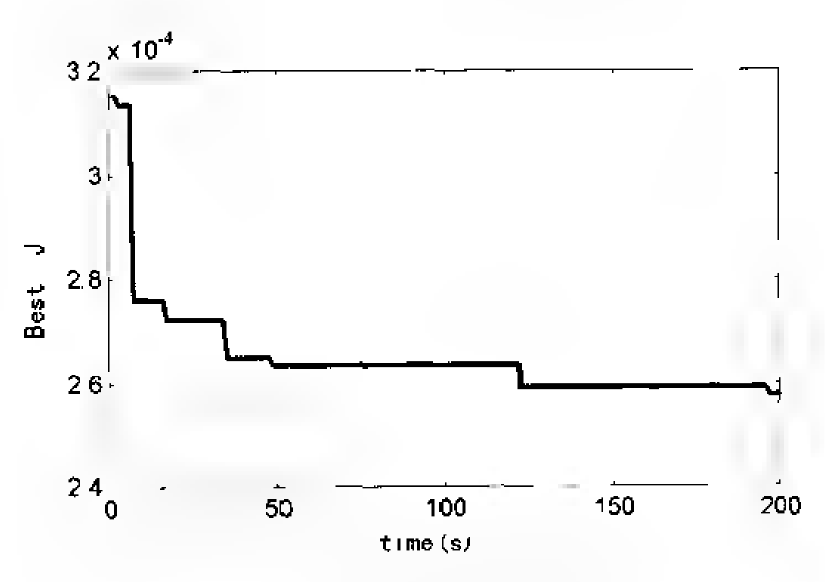


图 5-3　目标函数 J 的优化过程

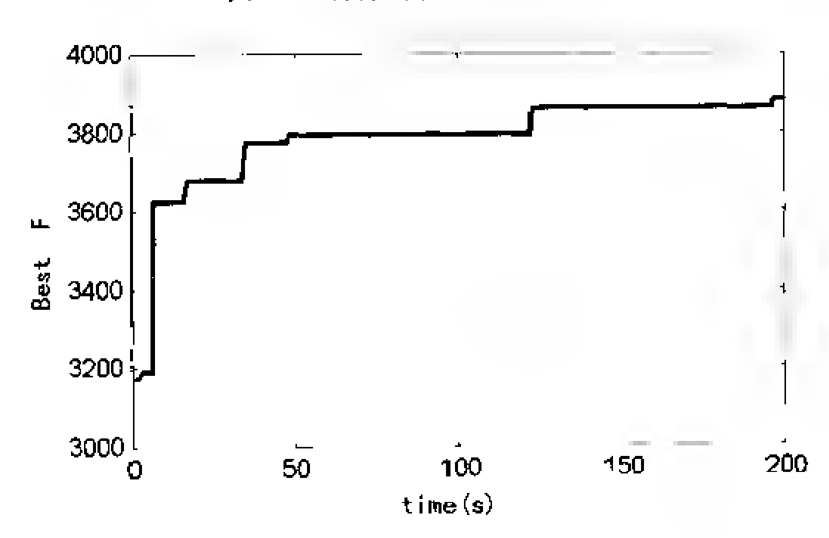


图 5-4　适应度函数 F 的优化过程

5.3.4 仿真程序

仿真程序：chap5_2.m

```
%Generic Algorithm for function f(x1,x2) optimum
clear all;
close all;

Size=500;
CodeL=2;

MinX(1)=-2.048;
MaxX(1)=2.048;
MinX(2)=-2.048;
MaxX(2)=2.048;

E(:,1)=MinX(1)+(MaxX(1)-MinX(1))*rand(Size,1);
E(:,2)=MinX(2)+(MaxX(2)-MinX(2))*rand(Size,1);

G=200;
BsJ=0;

%*************** Start Running ***************
for kg=1:1:G
time(kg)=kg;

%****** Step 1 : Evaluate BestJ ******
for i=1:1:Size
xi=E(i,:);
x1=xi(1);
x2=xi(2);

F(i)=100*(x1^2-x2)^2+(1-x1)^2;

Ji=1./F;
BsJi(i)=min(Ji);

end

[OderJi,IndexJi]=sort(BsJi);
BestJ(kg)=OderJi(1);
```

```
BJ-BestJ(kg);
Ji BsJi+1e-10;    %Avoiding deviding zero

fi=F;

   [Oderfi,Indexfi] sort(fi);    %Arranging fi small to bigger
   Bestfi=Oderfi(Size);          %Let Bestfi=max(fi)
   BestS=E(Indexfi(Size),:);    %Let BestS=E(m), m is the Indexfi belong to
max(fi)

   bfi(kg) Bestfi;

   kg
   BestS
%****** Step 2 : Select and Reproduct Operation******
   fi_sum sum(fi);
   fi_Size (Oderfi/fi_sum)*Size;

   fi_S floor(fi_Size);                  % Selecting Bigger fi value
   r=Size-sum(fi_S);

   Rest fi_Size-fi_S;
   [RestValue,Index] sort(Rest);

   for i=Size:-1:Size-r+1
     fi_S(Index(i)) fi_S(Index(i))+1;    % Adding rest to equal Size
   end

   k 1;
   for i=Size:-1:1      % Select the Sizeth and Reproduce firstly
     for j 1:1:fi_S(i)
      TempE(k,:) E(Indexfi(i),:);     % Select and Reproduce
        k=k+1;                        % k is used to reproduce
     end
   end

%************ Step 3 : Crossover Operation ************
    Pc=0.90;
    for i 1:2:(Size-1)
          temp rand;
```

```
      if temp>Pc                  %Crossover Condition
        alfa=rand;
        TempE(i,:)=alfa*E(i+1,:)+(1-alfa)*E(i,:);
        TempE(i+1,:)=alfa*E(i,:)+(1-alfa)*E(i+1,:);
      end
    end
    TempE(Size,:)=BestS;
    E=TempE;

%************* Step 4 Mutation Operation **************
Pm=0.10-[1:1:Size]*(0.01)/Size;       %Bigger fi,smaller Pm
Pm_rand=rand(Size,CodeL);
Mean=(MaxX + MinX)/2;
Dif=(MaxX-MinX);

   for i=1:1:Size
      for j=1:1:CodeL
         if Pm(i)>Pm_rand(i,j)        %Mutation Condition
            TempE(i,j)=Mean(j)+Dif(j)*(rand-0.5);
         end
      end
   end
%Guarantee TempE(Size,:) belong to the best individual
   TempE(Size,:)=BestS;
   E=TempE;
end
BestS
Bestfi

figure(1);
plot(time,BestJ,'k');
xlabel('Times');ylabel('Best J');

figure(2);
plot(time,bfi,'k');
xlabel('times');ylabel('Best F');
```

5.4 基于遗传算法的 PID 整定

PID 控制是工业过程控制中应用最广的策略之一，因此 PID 控制器参数的优化成为人们

关注的问题，它直接影响控制效果的好坏，并和系统的安全、经济运行有着密不可分的关系。目前PID参数的优化方法有很多，如间接寻优法，梯度法，爬山法等，而在热工系统中单纯形法、专家整定法则应用较广。虽然这些方法都具有良好的寻优特性，但存在着一些弊端，单纯形法对初值比较敏感，容易陷入局部最优化解，造成寻优失败。专家整定法则需要太多的经验，不同的目标函数对应不同的经验，而整理知识库则是一项长时间的工程。因此我们选取了遗传算法来进行参数寻优，该方法是一种不需要任何初始信息并可以寻求全局最优解的、高效的优化组合方法。

采用遗传算法进行PID三个系数的整定，具有以下优点：

（1）与单纯形法相比，遗传算法同样具有良好的寻优特性，且它克服了单纯形法参数初值的敏感性。在初始条件选择不当的情况下，遗传算法在不需要给出调节器初始参数的情况下，仍能寻找到合适的参数，使控制目标满足要求。同时单纯形法难以解决多值函数问题以及在多参数寻优（如串级系统）中，容易造成寻优失败或时间过长，而遗传算法的特性决定了它能很好地克服以上问题。

（2）与专家整定法相比，它具有操作方便、速度快的优点，不需要复杂的规则，只通过字串进行简单地复制、交叉、变异，便可达到寻优。避免了专家整定法中前期大量的知识库整理工作及大量的仿真实验。

（3）遗传算法是从许多点开始并行操作，在解空间进行高效启发式搜索，克服了从单点出发的弊端及搜索的盲目性，从而使寻优速度更快，避免了过早陷入局部最优解。

（4）遗传算法不仅适用于单目标寻优，而且也适用于多目标寻优。根据不同的控制系统，针对一个或多个目标，遗传算法均能在规定的范围内寻找到合适参数。

遗传算法作为一种全局优化算法，得到越来越广泛的应用。近年来，遗传算法在控制上的应用日益增多。

5.4.1 基于遗传算法的PID整定原理

1. 参数的确定及表示

首先确定参数范围，该范围一般是由用户给定的，然后由精度的要求，对其进行编码。选取二进制字串来表示每一个参数，并建立与参数间的关系。再把二进制串连接起来就组成一个长的二进制字串，该字串为遗传算法可以操作的对象。

2. 选取初始种群

因为需要编程来实现各过程，所以采用计算机随机产生初始种群。针对二进制编码而言，先产生0～1之间均匀分布的随机数，然后规定产生的随机数0～0.5之间代表0，0.5～1之间代表1。此外，考虑到计算的复杂程度来规定种群的大小。

3. 适配函数的确定

一般的寻优方法在约束条件下可以求得满足条件的一组参数，在设计中是从该组参数中寻找一个最好的。衡量一个控制系统的指标有三个方面，即稳定性、准确性和快速性。而上

升时间反映了系统的快速性，上升时间越短，控制进行得就越快，系统品质也就越好。

如果单纯追求系统的动态特性，得到的参数很可能使控制信号过大，在实际应用中会因系统中固有的饱和特性而导致系统不稳定，为了防止控制能量过大，在目标函数中加入控制量。因此为了使控制效果更好，我们给出了控制量、误差和上升时间作为约束条件。因为适应函数同目标函数相关，所以目标函数确定后，直接将其作为适配函数进行参数寻优。最优的控制参数也就是在满足约束条件下使 $f(x)$ 最大时，x 所对应的控制器参数。

4. 遗传算法的操作

首先利用适应度比例法进行复制，即通过适配函数求得适配值，进而求每个串对应的复制概率。复制概率与每代字串的个数的乘积为该串在下一代中应复制的个数。复制概率大的在下一代中将有较多的子孙，相反则会被淘汰。

其次进行单点交叉，交叉概率为 P_c。从复制后的成员里以 P_c 的概率选取字串组成匹配池，而后对匹配池的成员随机匹配，交叉的位置也是随机确定的。

最后以概率 P_m 进行变异。假如每代有 15 个字串，每个字串 12 位，则共有 $15\times12=180$ 个串位，期望的变异串位数为 $180\times0.01\approx2$ 位，即每代中有两个串位要由 1 变为 0 或由 0 变为 1。

初始种群通过复制、交叉及变异得到了新一代种群，该代种群经解码后代入适配函数，观察是否满足结束条件，若不满足，则重复以上操作，直到满足为止。

结束条件由具体问题所定，只要各目标参数在规定范围内，则终止计算。

以上操作过程可以用图 5-5 来表示。

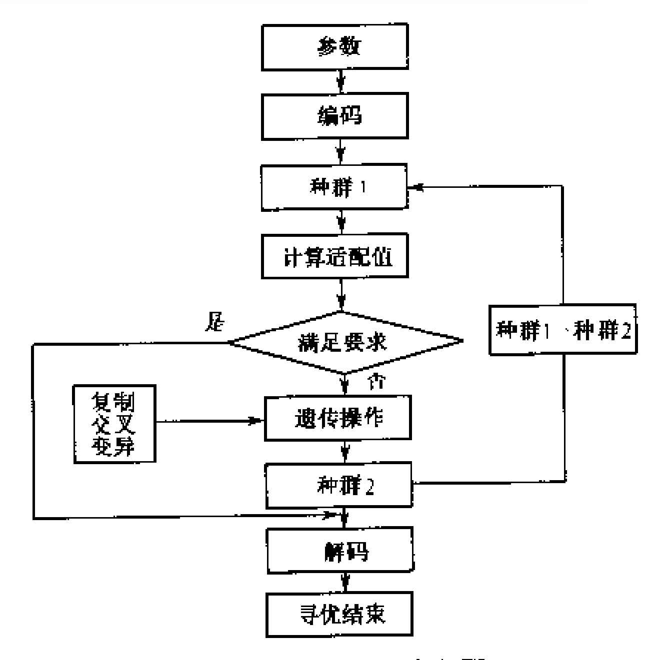


图 5-5　遗传算法流程图

利用遗传算法优化 k_p, k_i, k_d 的具体步骤如下：

（1）确定每个参数的大致范围和编码长度，进行编码；

（2）随机产生 n 个个体构成初始种群 $P(0)$；

（3）将种群中各个体解码成对应的参数值，用此参数求代价函数值 J 及适应函数值 f，取 $f=\frac{1}{J}$；

（4）应用复制、交叉和变异算子对种群 $P(t)$ 进行操作，产生下一代种群 $P(t+1)$；

（5）重复步骤（3）和（4），直至参数收敛或达到预定的指标。

5.4.2　基于实数编码遗传算法的PID整定

被控对象为二阶传递函数：

$$G(s)=\frac{400}{s^2+50s}$$

采样时间为1ms，输入指令为一阶跃信号。

为获取满意的过渡过程动态特性，采用误差绝对值时间积分性能指标作为参数选择的最小目标函数。为了防止控制能量过大，在目标函数中加入控制输入的平方项。选用下式作为参数选取的最优指标：

$$J=\int_0^{\infty}(w_1|e(t)|+w_2u^2(t))\mathrm{d}t+w_3\cdot t_u \tag{5.6}$$

式中，$e(t)$ 为系统误差，$u(t)$ 为控制器输出，t_u 为上升时间，w_1,w_2,w_3 为权值。

为了避免超调，采用了惩罚功能，即一旦产生超调，将超调量作为最优指标的一项，此时最优指标为：

$$\text{if } \mathrm{ey}(t)<0 \quad J=\int_0^{\infty}(w_1|e(t)|+w_2u^2(t)+w_4|\mathrm{ey}(t)|)\mathrm{d}t+w_3\cdot t_u \tag{5.7}$$

式中，w_3 为权值，且 $w_4 >> w_1$，$\mathrm{ey}(t)=y(t)-y(t-1)$，$y(t)$ 为被控对象输出。

遗传算法中使用的样本个数为30，交叉概率和变异概率分别为：$P_c=0.9, P_m=0.033$。参数 k_p 的取值范围为[0,20]，k_i, k_d 的取值范围为[0,1]，取 $w_1=0.999, w_2=0.001, w_4=100, w_3=2.0$。采用实数编码方式，经过100代进化，获得的优化参数如下：PID整定结果为 $k_p=19.0823, k_d=0.2434, k_i=0.0089$，性能指标 $J=23.9936$。代价函数 J 的优化过程和采用整定后的PID控制阶跃响应如图5.6和图5.7所示。

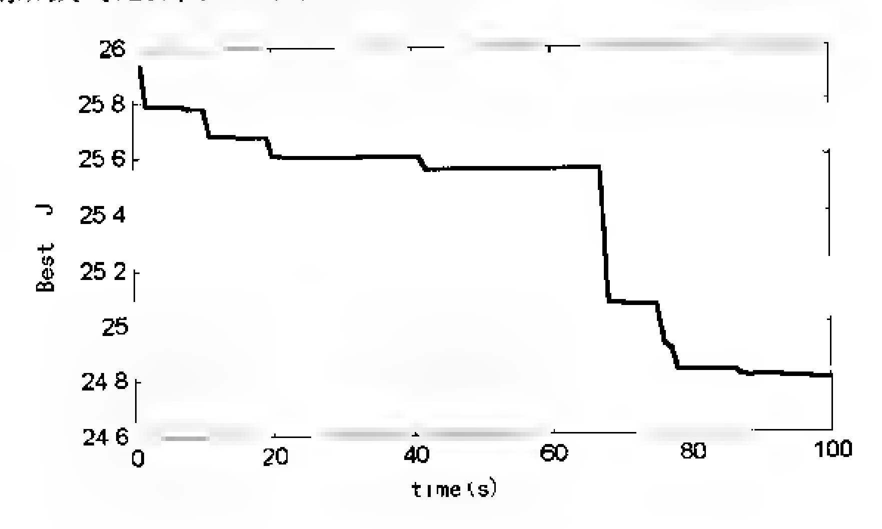


图5.6　代价函数 J 的优化过程

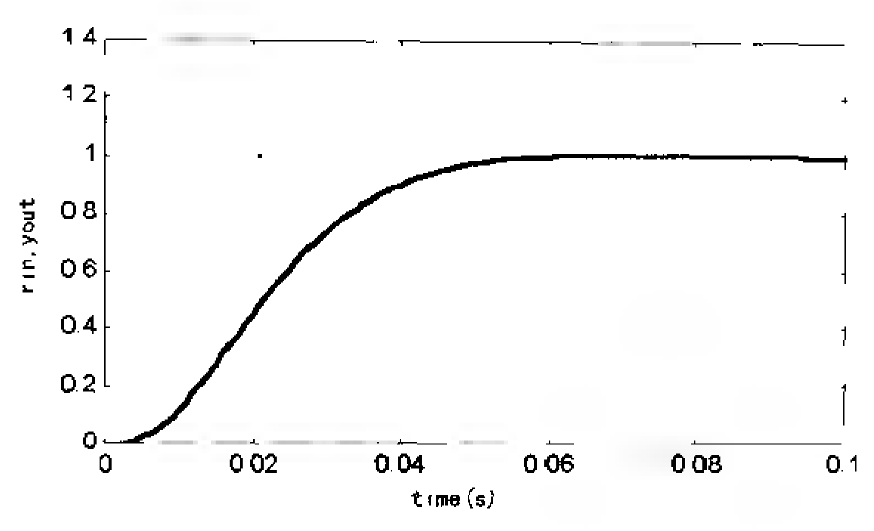


图 5-7　整定后的 PID 阶跃响应

在应用遗传算法时，为了避免参数选取范围过大，可以先按经验选取一组参数，然后在这组参数的周围利用遗传算法进行设计，从而大大减小初始寻优的盲目性，节约计算量。

5.4.3　仿真程序

主程序：chap5_3.m。

```
%GA(Generic Algorithm) Program to optimize PID Parameters
clear all;
close all;
global rin yout timef

Size=30;
CodeL=3;

MinX(1)=zeros(1);
MaxX(1)=20*ones(1);
MinX(2)=zeros(1);
MaxX(2)=1.0*ones(1);
MinX(3)=zeros(1);
MaxX(3)=1.0*ones(1);

Kpid(:,1)=MinX(1)+(MaxX(1)-MinX(1))*rand(Size,1);
Kpid(:,2)=MinX(2)+(MaxX(2)-MinX(2))*rand(Size,1);
Kpid(:,3)=MinX(3)+(MaxX(3)-MinX(3))*rand(Size,1);

G=100;
BsJ=0;
```

```
%*************** Start Running ***************
for kg 1:1:G
    time kg -kg;

%****** Step 1 : Evaluate BestJ ******
for i 1:1:Size
Kpidi Kpid(i,:);

[Kpidi,BsJ]-chap5 of(Kpidi,BsJ);

BsJi(i) BsJ;
end

[OderJi,IndexJi] sort(BsJi);
BestJ kg  OderJ( 1),
BJ-BestJ kg ;
Ji-BsJi+1e 10;    %Avoiding deviding zero

   fi 1./Ji;
%  Cm max(Ji ;
%  fi Cm Ji,

   [Oderfi,Indexfi] sort(fi);    %Arranging fi small to bigger
   Bestfi Oderfi Size ;          %Let Bestfi max(fi)
   BestS Kpid(Indexfi(Size),:);  %Let BestS-E m), m is the Indexfi belong to
max(fi

   kg
   BJ
   BestS
%****** Step 2 : Select and Reproduct Operation******
   fi_sum-sum fi);
   fi_Size (Oderfi fi sum)*Size;

   fi S-floor(fi Size);                  % Selecting Bigger fi value
   r Size-sum(fi S);

   Rest fi_Size fi_S;
   [RestValue Index] sort(Rest);

   for i Size. 1:Size-r+1
```

```
    [. S(Index ) , f1..](Iraex 1   +1;      * Addirg rest to equal Size
  end

  k l;
  for 1 ..ze: 1.]        * Select the Size'n and Reprodjce firstly
    for j ]:l:f1_S 1
     TempE k,·, Kpid Indexf1 1),: ;        % Select and Reproduce
       k k+1;                         % k is .sed to reprod.ce
    end
  end

%************ S.ep 3 : Cross ver Opera.. .. ************
   Pc 0 90;
   for 1 1·2: S.ze 1)
        temp rand;
     .f Pc ·temp                    % rossover Cond.tion
        alfa-rand;
        TempE(1,:) al.a*Kpid 1+1,: + 1 alfa *Kpid(i,:,;
        TempE.1+1,: -a'fa*Kpid 1,  + 1 alfa *Kpid(.+1,:);
     end
   end
   TempE S.ze,:  BestS,
   Kp.d TempE;

%************ Step 4: M.tation Operation **************
Pm-0.10 [1:1:S.ze]*(0.01) S.ze;       %Bigger f1,smaller Pm
Pm_rand rand S1ze,Code' ;
Mean  MaxX + M.nX /2;
D.f  (MaxX M1nX ;

   for 1 1:1:Size
     for j 1:1 CodeL
        if Pm 1 >Pm_rand(1,1)        %Mutation Condition
          TempE.1,j, Mean 1 +Dif()]*(rand 0.5);
        end
     end
   end
%Guarantee TempE S.ze,:  belong to the best individual
   TempE(Size,    BestS;
   Kpid TempE,
end
Bestf1
```

```
Best_S
Best_J=BestJ(G)
figure(1);
plot(time,BestJ);
xlabel('Times');ylabel('Best J');
figure(2);
plot(timef,rin,'r',timef,yout,'b');
xlabel('Time(s)');ylabel('rin,yout');
```

子程序：chap5_3f.m

```
function [Kpidi,BsJ]=pid_gaf(Kpidi,BsJ)
global rin yout timef

ts=0.001;
sys=tf(400,[1,50,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

rin=1.0;
u_1=0.0;u_2=0.0;
y_1=0.0;y_2=0.0;
x=[0,0,0]';
B=0;
error_1=0;
tu=1;
s=0;
P=100;

for k=1:1:P
   timef(k)=k*ts;
   r(k)=rin;

   u(k)=Kpidi(1)*x(1)+Kpidi(2)*x(2)+Kpidi(3)*x(3);

   if u(k)>=10
      u(k)=10;
   end
   if u(k)<=-10
      u(k)=-10;
   end

   yout(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;
```

```
   error(k)=r(k)-yout(k);
%------------ Return of PID parameters ------------
   u_2=u_1;u_1=u(k);
   y_2=y_1;y_1=yout(k);

   x(1)=error(k);                  % Calculating P
   x(2)=(error(k)-error_1)/ts;    % Calculating D
   x(3)=x(3)+error(k)*ts;         % Calculating I

   error_2=error_1;
   error_1=error(k);
if s==0
   if yout(k)>0.95&yout(k)<1.05
      tu=timef(k);
      s=1;
   end
end
end

for i=1:1:P
   Ji(i)=0.999*abs(error(i))+0.01*u(i)^2*0.1;
   B=B+Ji(i);
  if i>1
   erry(i)=yout(i)-yout(i-1);
   if erry(i)<0
      B=B+100*abs(erry(i));
   end
  end
end
BsJ=B+0.2*tu*10;
```

5.4.4 基于二进制编码遗传算法的PID整定

被控制对象为二阶传递函数：

$$G(s)=\frac{400}{s^2+50s}$$

采样时间为1ms，输入指令为阶跃信号。

采用二进制编码方式，用长度为10位的二进制编码串分别表示三个决策变量k_p, k_i, k_d。

最优指标的选取同十进制编码遗传算法的PID整定。遗传算法中使用的样本个数为Size = 30，交叉概率和变异概率分别为：$P_c = 0.60, P_m = 0.001-[1:1:\text{Size}]\times 0.001/\text{Size}$。

参数k_p的取值范围为[0,20]，k_i, k_d的取值范围为[0,1]，w_1, w_2, w_3, w_4的取值同十进制编码遗传算法的PID整定。经过100代进化，获得的优化参数如下：

最优个体为 BestS=[010110111111011000100010000100]。PID 优化参数为：$k_p=16.1290, k_d=0.2209, k_i=0.2209$，性能指标 $J=24.9812$、整定过程中代价函数 J 的变化如图 5-8 所示。采用整定后的二进制遗传算法优化 PID 阶跃响应如图 5-9 所示。

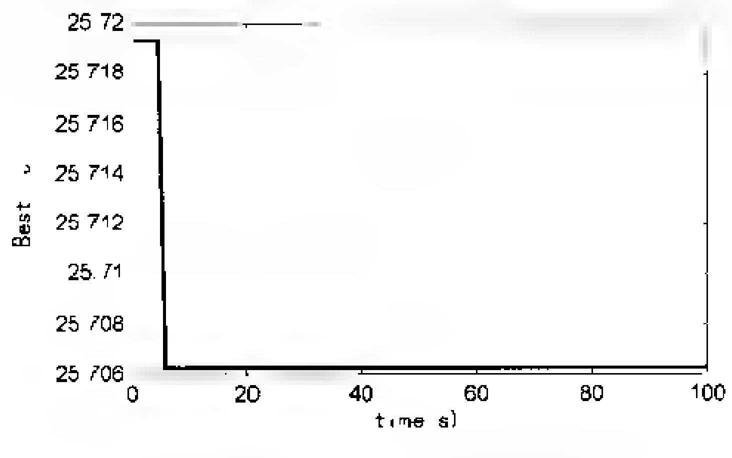


图 5-8 代价函数 J 的优化过程

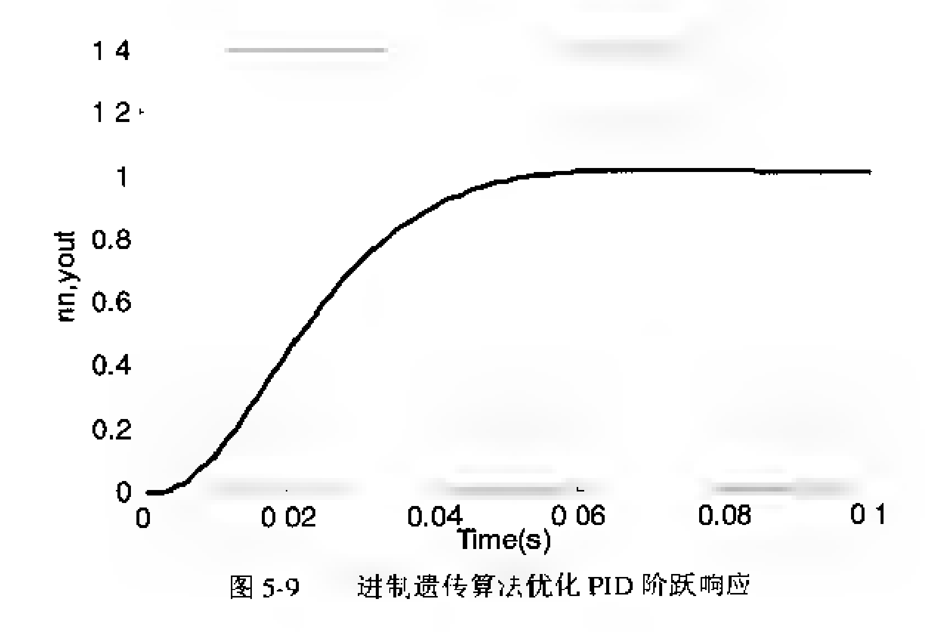


图 5-9 二进制遗传算法优化 PID 阶跃响应

5.4.5 仿真程序

主程序：chap5_4.m。

```
%GA(Generic Algorithm) Program to optimize Parameters of PID
clear all;
close all;
global rin yout timef

G=100;
Size=30;
CodeL=10;
```

```
MinX(1)=zeros(1);
MaxX(1)=20*ones(1);
MinX(2)=zeros(1);
MaxX(2)=1.0*ones(1);
MinX(3)=zeros(1);
MaxX(3)=1.0*ones(1);

E=round(rand(Size,3*CodeL));    %Initial Code!

BsJ=0;

for kg=1:1:G
time(kg)=kg;

for s=1:1:Size
m=E(s,:);
y1=0;y2=0;y3=0;

m1=m(1:1:CodeL);
for i=1:1:CodeL
   y1=y1+m1(i)*2^(i-1);
end
Kpid(s,1)=(MaxX(1)-MinX(1))*y1/1023+MinX(1);

m2=m(CodeL+1:1:2*CodeL);
for i=1:1:CodeL
   y2=y2+m2(i)*2^(i-1);
end
Kpid(s,2)=(MaxX(2)-MinX(2))*y2/1023+MinX(2);

m3=m(2*CodeL+1:1:3*CodeL);
for i=1:1:CodeL
   y3=y3+m3(i)*2^(i-1);
end
Kpid(s,3)=(MaxX(3)-MinX(3))*y3/1023+MinX(3);

%****** Step 1 : Evaluate BestJ ******
Kpidi=Kpid(s,:);

[Kpidi,BsJ]=chap5_3f(Kpidi,BsJ);
```

```
    BsJ1(s) BsJ;
    end

   [OderJ1,IndexJ1] sort BsJ1 ,
   BestJ kg  OderJ1(1);
   BJ BestJ kg);
   J1-BsJ1+1e 10;

     f1-1. J1;
   % Cm max J1 ;
   % fi Cm J1;        %Avoiding deviding zero

     [Oderf1,Indexf1] sort(f1);         %Arranging f1 small to bigger
   %  Bestf1 Oderf1(Size);              %Let Bestf1 max f1)
   %  BestS Kpid(Indexf1 Size ,:);     %Let BestJ E m , m is the Indexf1 belong
to max f1)

   Bestf1-Oderf1(Size);         % Let Bestf1 max f1
   BestS-E(Indexf1 Size,,  ;   % Let BestS E m , m is the Indexf1 belong to max f1

   kg
   BJ
   BestS;

   %****** Step 2 : Select and Reproduct Operation******
     f1 sum-sum(f1 ;
     f1_Size- Oderf1 f1 sum *Size;

     f1_S-floor(f1 Size);       %Selecting Bigger f1 value

     kk 1;
     for i 1:1:Size
       for j-1:1:f1_S i)       %Select and Reproduce
        TempE kk,:)-E(Indexf1(i ,:);
          kk kk+1;              %kk is sed to reproduce
       end
     end

   %************ Step 3 : Crossover Operation ************
   pc 0.60;
   n-ceil(20*rand ;
   for i 1:2:(Size 1
```

```
    temp rand;
    if pc>temp                  %Crossover Condition
    for j=n:1:20
        TempE(i,j)=E(i+1,j);
        TempE(i+1,j)=E(i,j);
    end
    end
end
TempE(Size,:)=BestS;
E=TempE;

%************ Step 4: Mutation Operation **************
%pm=0.001;
pm=0.001-[1:1:Size]*(0.001)/Size; %Bigger fi, smaller pm
%pm=0.0;     %No mutation
%pm=0.1;     %Big mutation

   for i=1:1:Size
      for j=1:1:3*CodeL
         temp=rand;
         if pm>temp              %Mutation Condition
            if TempE(i,j)==0
               TempE(i,j)=1;
            else
               TempE(i,j)=0;
            end
        end
      end
   end
%Guarantee TempE(Size,:) belong to the best individual
TempE(Size,:)=BestS;
E=TempE;
%**********************************************************
end

Bestfi
BestS
Kpid
Best_J=BestJ(G)
figure(1);
plot(time,BestJ);
xlabel('Times');ylabel('Best J');
```

```
figure(2);
plot(timef,rin,'r',timef,yout,'b');
xlabel('Time(s)');ylabel('rin,yout');
```

子程序：chap5_3f.m（同前）。

5.4.6 基于自适应在线遗传算法整定的PID控制

所谓在线PID整定，即在每个采样时间分别对PD参数进行整定。采用遗传算法在线整定PID，就是针对每个采样时间实现PID控制参数的遗传算法优化。在采样时间k，选取足够多的个体，计算不同个体的自适应度，通过遗传算法的优化，选择自适应度大的个体所对应的PD控制参数作为该采样时间下PD的控制参数。

设被控对象为二阶传递函数：

$$G(s) = \frac{400}{s^2 + 50s}$$

采样时间为1ms，输入指令为一阶跃信号。

为获取满意的过渡过程动态特性，并防止产生超调，采用误差绝对值、误差和误差变化率的加权及作为第k个采样时间时第i个个体的参数选择最小目标函数。

$$J(i) = \alpha_p \times |\mathrm{errori}(i)| + \beta_p \times |\mathrm{de}(i)| \tag{5.8}$$

式中，errori(i)为第k个采样时间第i个个体的位置跟踪误差，de(i)为第k个采样时间第i个个体的位置跟踪误差变化率。

为了避免超调，采用了惩罚功能，即一旦产生超调，将超调量作为最优指标的一项，此时最优指标为：

$$\text{if errori}(i)<0 \quad J(i) = J(i) + 100|\mathrm{errori}(i)| \tag{5.9}$$

针对每个采样时间进行PD参数的遗传算法优化。在仿真程序中，M=1时为采用遗传算法，否则为未整定的PID控制。在最小目标函数中，取$\alpha_p = 0.95$，$\beta_p = 0.05$。遗传算法中使用的个体数为120，进化代数为10代。交叉概率为$P_c = 0.9$，采用自适应变异概率方法，即自适应度越大，变异概率越小，变异概率为$P_m = 0.20 - [1:1:\mathrm{Size}] \times 0.01/\mathrm{Size}$。采用实数编码方式，参数$k_p$的取值范围为$[9.0, 12.0]$，$k_d$的取值范围为$[0.2, 0.3]$。

PD的阶跃响应、PD整定过程中控制器$u(k)$的变化及参数k_p、k_d的整定过程如图5-10～图5-13所示。由仿真结果可见，在控制的初始阶段（误差小于0.50时），为了尽快降低误差，k_p上升、k_d下降；当上升到一定程度时（误差大于0.50时），为了防止误差变化太快而产生超调，k_p下降、k_d上升；当上升到指令值而产生超调时（t = 0.40s时），为了尽快降低误差，k_p上升、k_d下降。

为了避免参数选取范围过大，先按经验选取一组k_p、k_d参数，然后在这组参数的周围利用遗传算法进行设计，从而减少初始寻优的盲目性，节省计算量。

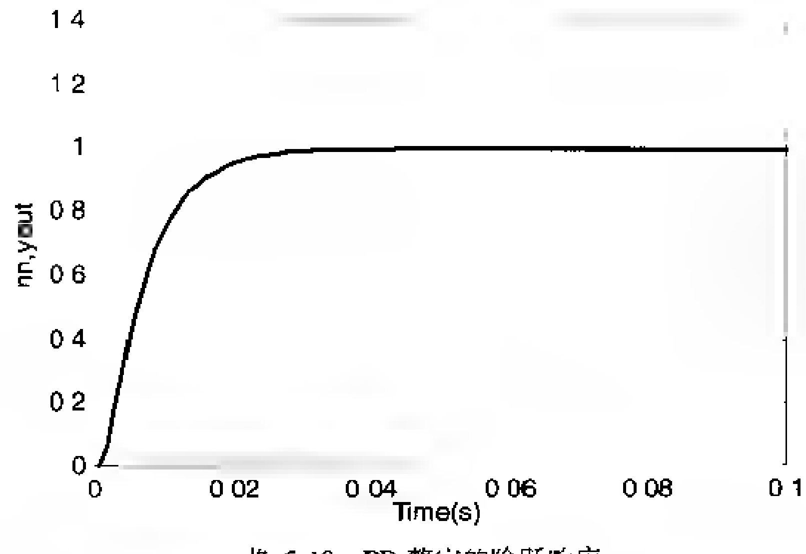


图 5-10 PD 整定的阶跃响应

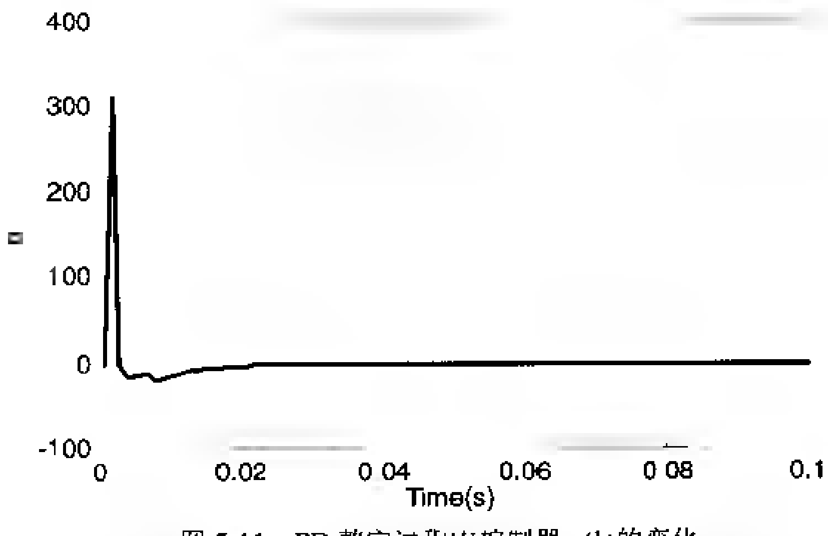


图 5-11 PD 整定过程中控制器 $u(k)$ 的变化

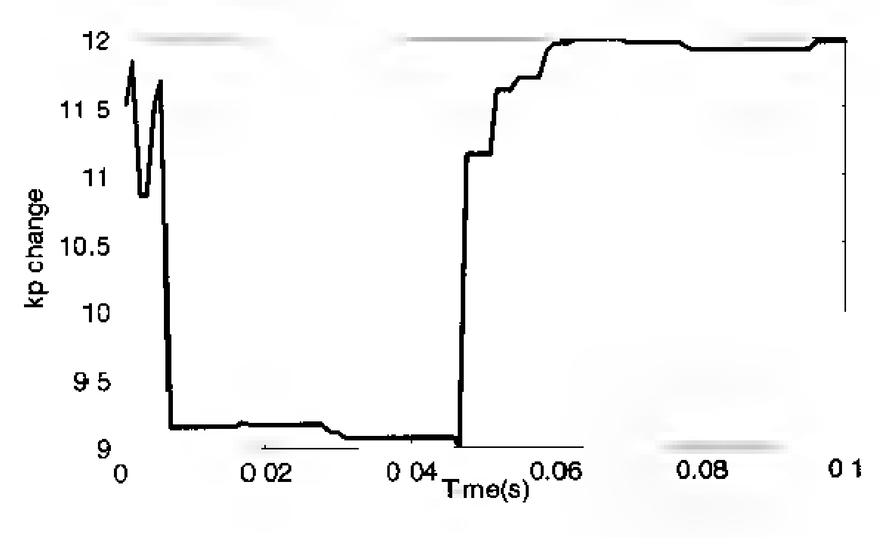


图 5-12 PD 整定过程中 k_p 的变化

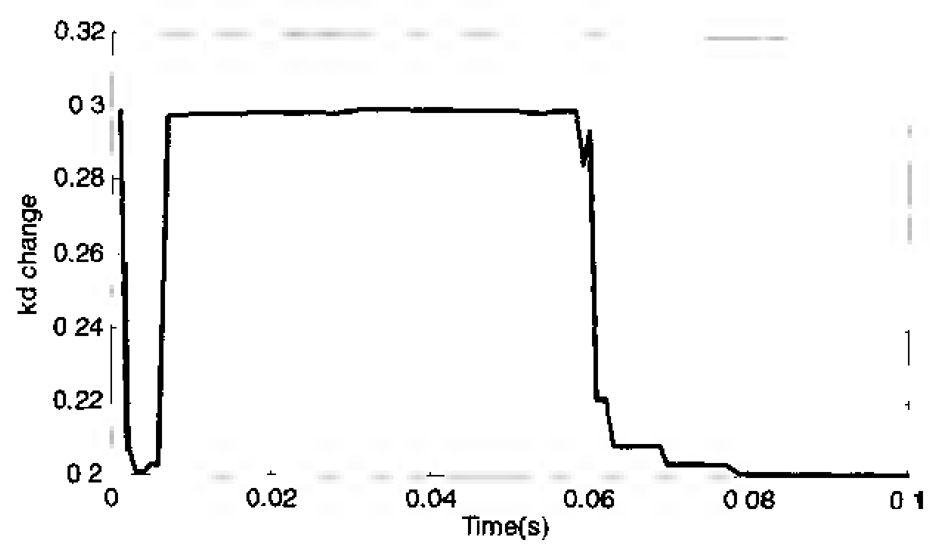


图 5-13　PD 整定过程中 k_d 的变化

5.4.7　仿真程序

仿真程序的主程序：chap5_5.m。

```
%GA(Generic Algorithm) to Optimize Online PID Control
clear all;
close all;

Size=120;
CodeL=2;

MinX(1)=9*ones(1);MaxX(1)=12*ones(1);
MinX(2)=0.20*ones(1);MaxX(2)=0.30*ones(1);

Kpid(:,1)=MinX(1)+(MaxX(1)-MinX(1))*rand(Size,1);
Kpid(:,2)=MinX(2)+(MaxX(2)-MinX(2))*rand(Size,1);

BsJ=0;
J=0;

x=zeros(1,2);
x1=zeros(1,2);
xk=zeros(1,2);
ts=0.001;

error_1=0;

BestS=zeros(2,1);
for k=1:1:100
time(k)=k*ts;
```

```
rin(k)=1;

%u(k)=10*x(1)+0.2*x(2);   %Test PI: good results

u(k)=BestS(1)*x(1)+BestS(2)*x(2);
para=u(k);
tSpan=[0 ts];

[tt,xx]=ode45('chap5_5f',tSpan,xk,[],para);
xk=xx(length(xx),:);
yout(k)=xk(1);

error(k)=rin(k)-yout(k);
x(1)=error(k);                    % Calculating P
x(2)=(error(k)-error_1)/ts;    % Calculating D


error_1=error(k);

B=0;

M=1;   %Using GA
if M==1


G=10;
for kg=1:1:G    %Era evolution

%****** Step 1 : Evaluate BestJ ******
for i=1:1:Size

Kpidi=Kpid(i,:);

u1(i)=Kpidi(1)*x(1)+Kpidi(2)*x(2);

para=u1(i);
[tt,xx1]=ode45('chap5_5f',tSpan,xk,[],para);
y1=xx1(length(xx1),1);

error1(i)=rin(k)-y1;
du(i)=u1(i)-u(k);
de(i)=(error1(i)-error(k))/ts;
alfap=0.95;
betap=0;

if abs(error(k))<=0.50
   betap=0.05;
```

```
end
J=alfap*abs(errori(i))+betap*abs(de(i));

B=J;

if errori(i)<0
   B=B+100*abs(errori(i));
end

BsJi(i)=B;

[OderJi,IndexJi]=sort(BsJi);
BestJ(kg)=OderJi(1);
BJ=BestJ(kg);
Ji=BsJi+1e-10;    %Avoiding deviding zero

fi=1./Ji;
%Cm=max(Ji);
%fi=Cm-Ji;
end    %End of a Size...!!....!

   [Oderfi,Indexfi]=sort(fi);    %Arranging fi small to bigger

   Bestfi=Oderfi(Size);          %Let Bestfi=max(fi)

   BestS=Kpid(Indexfi(Size),:);  %Let BestS=E(m), m is the Indexfi belong to
max(fi)

%****** Step 2 : Select and Reproduct Operation******
   fi_sum=sum(fi);
   fi_Size=(Oderfi/fi_sum)*Size;

   fi_S=floor(fi_Size);                     % Selecting Bigger fi value
   r=Size-sum(fi_S);

   Rest=fi_Size-fi_S;
   [RestValue,Index]=sort(Rest);

   for i=Size:-1:Size-r+1

      fi_S(Index(i))=fi_S(Index(i))+1;    % Adding rest to equal Size
   end

   k=1;
```

```
  for i-Size: 1:1       % Select the Sizcth and Reproduce firstly
     for j-1:1:fi_S(i)
      TempE(kr,:)-Kpid(Indexfi(i),:);        % Select and Reproduce
        kr kr+1,                          % kr is used to reproduce
     end
  end

%************ Ctep 3 : Crossover Operation ************
   Pc-0.90;
   for i-1:2  Size 1
         temp-rand,
     if Pc>temp                    %Crossover Condition
         alfa rand,
         TempE(i,:)-alfa*Kpid i+1,: +(1 alfa,*Kpid i,: ;
         TempE(i+1,: -alfa*Kpid(i,:,+(1 alfa)*Kpid(i+1,:);
     end
   end
   TempE(Size,:)-BestS;
   Kpid TempE;

%************ Step 4: Mutation Operation *************
Pm-0.20-[1:1:Size]* 0.01)/Size;       %Bigger fi,smaller Pm
im_rand rand(Size,CodeL);
Mean (MaxX + MinX)/2;
Dif  MaxX MinX ;

  for i-1:1:Size
     for   1:1:CodeL
        if Pm(i)>Pm rand(i,j)        %Mutation Condition
           TempE(i,j)-Mean(j +Dif(j)*(rand 0.5);
        end
     end
  end
%Guarantee TempE(Size,:) belong to the best individual
TempE(Size, ) -BestS;
Kpid TempE;
end  %End of kg

     kph(k  BestS(1);
     kdh(k -BestS(2);
     BestS
```

```
end    %End of M 1

end    %End of k

figure(1);
plot(time,rin,'r',time,yout,'b');
xlabel('time(s)');ylabel('rin,yout');

figure(2);
plot(time,u,'r');
xlabel('Time(s)');ylabel('u');

if M==1
    figure(3);
    plot(time,kph,'r');
    xlabel('Time(s)');ylabel('kp change');
    figure(4);
    plot(time,kdh,'r');
    xlabel('Time(s)');ylabel('kd change');
end
```

被控对象子程序：chap5_5f.m。

```
function dx=PlantModel(t,x,flag,para)

dx=zeros(2,1);

u=para;

dx(1)=x(2);
dx(2)=-50*x(2)+400*u;
```

5.5 基于遗传算法摩擦模型参数辨识的 PID 控制

5.5.1 辨识原理及仿真实例

被控对象为二阶传递函数：

$$G(s)=\frac{400}{s^2+50s}$$

设外加在控制器输出上的干扰为一等效摩擦。

当 $F=1$ 时为库仑摩擦，摩擦模型为：

$$F_f(t)=0.8\mathrm{sgn}(\dot{\theta}(t)) \tag{5.10}$$

当 $F=2$ 时为库仑摩擦+粘性摩擦，摩擦模型为：

$$F_f(t)=\mathrm{sgn}(\dot{\theta}(t))(kx_1|\dot{\theta}(t)|+kx_2)=\mathrm{sgn}(\theta(t))(0.30|\theta(t)|+1.50) \tag{5.11}$$

式中，kx_1 和 kx_2 为待辨识参数。

采样时间为 1ms，取 $S=2$，使输入指令为阶跃信号。

为获取满意的过渡过程动态特性，采用误差绝对值时间积分性能指标作为参数选择的最小目标函数。为了防止控制能量过大，在目标函数中加入控制输入的平方项。选用下式作为参数选取的最优指标：

$$J=\int_0^{\infty}(w_1|e(t)|+w_2u^2(t))\mathrm{d}t \tag{5.12}$$

式中，$e(t)$ 为系统误差，w_1 和 w_2 为权值。

为了避免超调，采用了惩罚功能，即一旦产生超调，将超调量作为最优指标的一项，此时最优指标为：

$$\text{if } e(t)<0 \quad J=\int_0^{\infty}(w_1|e(t)|+w_2u^2(t)+w_3|e(t)|)\mathrm{d}t \tag{5.13}$$

式中，w_3 为权值，且 $w_3 \gg w_1$。

在应用遗传算法时，为了避免参数选取范围过大，可以先按经验选取一组参数，然后在这组参数的周围利用遗传算法进行设计，从而大大减小初始寻优的盲目性，节约计算量。

采用实数编码方式，遗传算法中使用的样本个数为 30，交叉概率和变异概率分别为：$P_c=0.9$，$P_m=0.10-[1:1:\mathrm{Size}]\times0.01/\mathrm{Size}$，$w_1=0.999$，$w_2=0.001$，$w_3=10$。

通过取 $kx=[0,0]$ 使摩擦补偿 $F_{fc}=0$，得到在无摩擦补偿情况下的阶跃响应如图 5-14 所示。

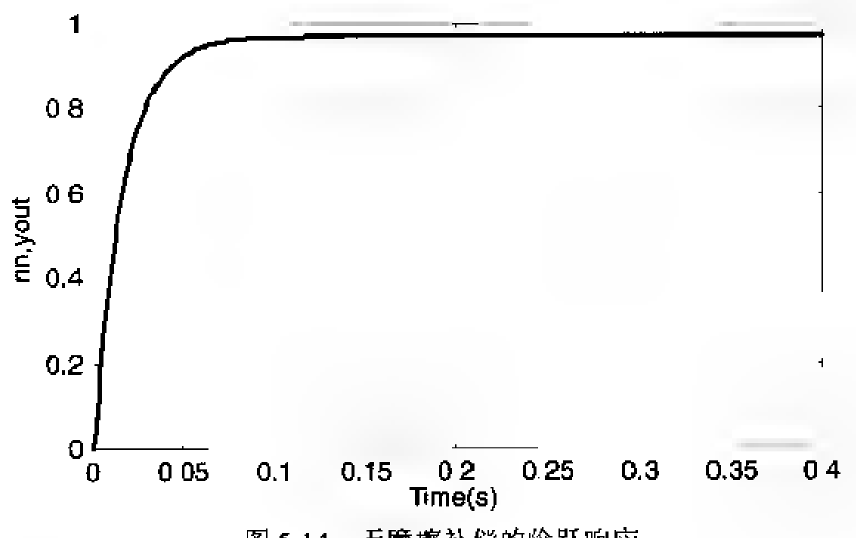


图 5-14 无摩擦补偿的阶跃响应

取 $F=2$，运行程序 chap5_6.m，采用遗传算法对摩擦模型进行辨识。待辨识参数采用实数编码法，取 $kx=[0.3,1.5]$，辨识参数 kx_1 和 kx_2 的范围选为 [0,2.0]，取进化代数为 50。经过优化获得的最优样本和最优指标为：BestS = [0.3523,1.2257]，Best_J = 25.5439。摩擦参数辨识结果为 $kx_1=0.3523, kx_2=1.2257$，采用摩擦补偿后的 PID 控制阶跃响应如图 5-15 所示，代价函数值 J 的优化过程如图 5-16 所示。

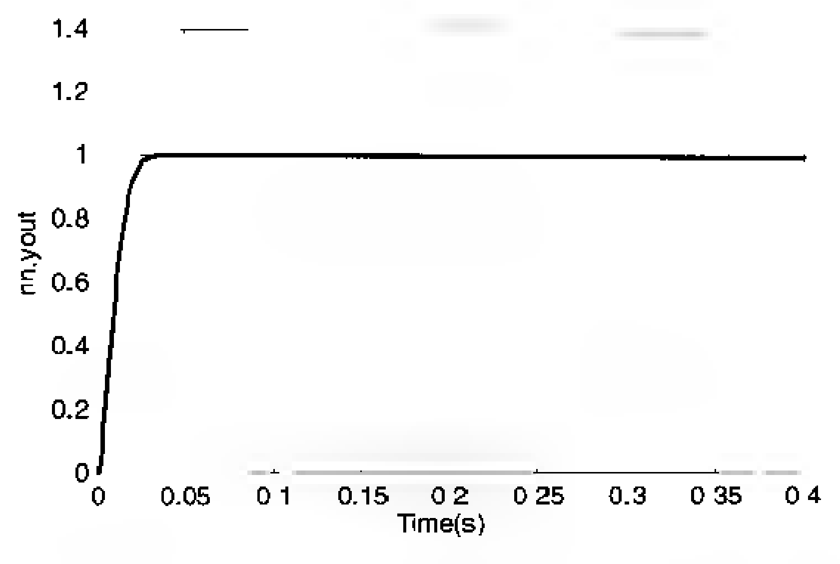


图 5-15　采用摩擦补偿后的 PID 控制阶跃响应

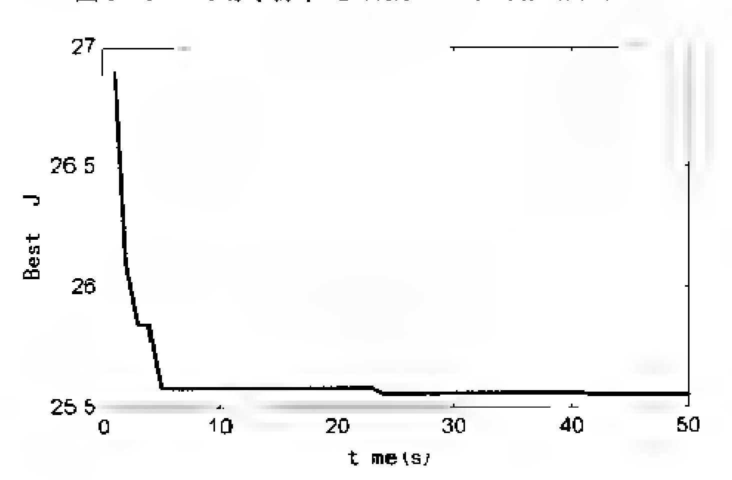


图 5-16　代价函数值 J 的优化过程

5.5.2　仿真程序

主程序为遗传算法程序，子程序为带有摩擦模型的 PID 控制程序。

主程序: chap5_6.m。

```
%GA(Generic Algorithm) Program to predict Parameters of Friction
clear all;
close all;
global rin yout timef F

Size=30;
F=2;
if F==1
   CodeL=1;
   MinX=zeros(CodeL,1);
   MaxX=1.0*ones(CodeL,1);
```

```
end
if F==2
   CodeL=2;
   MinX=zeros(CodeL,1);
   MaxX=2.0*ones(CodeL,1);
end

for i=1:1:CodeL
    kxi(:,i)=MinX(i)+(MaxX(i)-MinX(i))*rand(Size,1);
end

G=50;
BsJ=0;

for kg=1:1:G
    time(kg)=kg;

%****** Step 1:Evaluate BestJ ******
for i=1:1:Size
    kx=kxi(i,:);

[kx,BsJ]=chap5_6f(kx,BsJ);

BsJi(i)=BsJ;
end

[OderJi,IndexJi]=sort(BsJi);
BestJ(kg)=OderJi(1);
BJ=BestJ(kg);
Ji=BsJi+1e-10;

   fi=1./Ji;
%  Cm=max(Ji);
%  fi=Cm-Ji;                    %Avoiding deviding zero

   [Oderfi,Indexfi]=sort(fi);     %Arranging fi small to bigger
   Bestfi=Oderfi(Size);           %Let Bestfi=max(fi)
%Let BestS=E(m), m is the Indexfi belong to max(fi)
   BestS=kxi(Indexfi(Size),:);

   kg
```

```
   BJ
   BestS
   kx
%****** Step 2 : Select and Reproduct Operation******
   fi_sum=sum(fi);
   fi_Size=(Oderfi/fi_sum)*Size;

   fi_S=floor(fi_Size);                %Selecting Bigger fi value
   r_Size=sum(fi_S);

   Rest=fi_Size-fi_S;
   [RestValue,Index]=sort(Rest);

   for i=Size:-1:Size-r_Size+1
      fi_S(Index(i))=fi_S(Index(i))+1; %Adding rest to equal Size
   end

   k=1;
   for i=Size:-1:1          %Select the Sizeth and Reproduce firstly
     for j=1:1:fi_S(i)       %Notice: If i=1:1:Size then k plus meaningless
      TempE(k,:)=kxi(Indexfi(i),:);    %Select and Reproduce
        k=k+1;                          %k is used to reproduce
     end
   end

%************ Step 3 : Crossover Operation ************
    Pc=0.90;
    for i=1:2:(Size-1)
          temp=rand;
       if Pc>temp                         %Crossover Condition
          alfa=rand;
          TempE(i,:)=alfa*kxi(i+1,:)+(1-alfa)*kxi(i,:);
          TempE(i+1,:)=alfa*kxi(i,:)+(1-alfa)*kxi(i+1,:);
       end
    end
    TempE(Size,:)=BestS;
    kxi=TempE;

%************ Step 4: Mutation Operation **************
Pm=0.10-[1:1:Size]*(0.01)/Size;          %Bigger fi, smaller Pm
Pm_rand=rand(Size,CodeL);
Mean=(MaxX + MinX)/2;
```

```
Dif=(MaxX-MinX);

   for i=1:1:Size
      for j=1:1:CodeL
         if Pm(i)>Pm_rand(i,j)         %Mutation Condition
            TempE(i,j)=Mean(j)+Dif(j)*(rand-0.5);
         end
      end
   end
%Guarantee TempE(Size,:) belong to the best individual
   TempE(Size,:)=BestS;
   kxi=TempE;
%**************************************************************
end
Bestfi
BestS
Best_J=BestJ(G)

figure(1);
plot(timef,rin,'b',timef,yout,'r');
xlabel('Time(s)');ylabel('rin,yout');
figure(2);
plot(time,BestJ,'r');
xlabel('Times');ylabel('Best J');
```

子程序: chap5_6f.m。

```
function [kx,BsJ]=pid_fm_gaf(kx,BsJ)
global rin yout timef F

a=50;b=400;
ts=0.001;
sys=tf(b,[1,a,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

u_1=0;u_2=0;
y_1=0;y_2=0;
e_1=0;
B=0;

G=400;
for k=1:1:G
```

```
    timef(k)=k*ts;
S=2;
if S==1
    fre=5;
    AA=0.5;
    rin(k)=AA*sin(2*pi*fre*k*ts);
end
if S==2
    rin(k)=1;
end

yout(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;
error(k)=rin(k)-yout(k);
derror(k)=(error(k)-e_1)/ts;

u(k)=50*error(k)+0.50*derror(k);

speed(k)=(yout(k)-y_1)/ts;

if F==1 % Disturbance Signal: Coulomb Friction
    Ff(k)=0.8*sign(speed(k));
end
if F==2 % Disturbance Signal: Coulomb & Viscous Friction
    Ff(k)=sign(speed(k))*(0.30*abs(speed(k))+1.50);
end

%kx=[0,0];       %No GA Identification
%kx=[0.3,1.5];   %Idea Identification

u(k)=u(k)-Ff(k);

if F==1
    Ffc(k)=kx*sign(speed(k)); %Friction Estimation
end
if F==2  %Friction Estimation
    Ffc(k)=sign(speed(k))*(kx(1)*abs(speed(k))+kx(2));
end

u(k)=u(k)+Ffc(k);

if u(k)>=110
    u(k)=110;
```

```
end
if u(k)<-110
   u(k)=-110;
end

u_2=u_1;u_1=u(k);
y_2=y_1;y_1=yout(k);
e_1=error(k);
end
for i=1:1:G
   Ji(i)=0.999*abs(error(i))+0.01*u(i)^2*0.1;
   B=B+Ji(i);
   if error(i)<0    %Punishment
      B=B+10*abs(error(i));
   end
end
BsJ=B;
```

第6章　先进PID多变量控制

6.1　PID多变量控制

6.1.1　PID控制原理

通过PID控制，可实现多变量控制，图6-1给出一个二变量PID控制系统框图，该系统由两个PID控制器构成，控制算法为：

$$
\begin{aligned}
u_1(k) &= k_{p1}\text{error}_1(k) + k_{d1}\frac{\text{error}_1(k)-\text{error}_1(k-1)}{T} + k_{i1}\sum_{i=1}^{k}\text{error}_1(i)T \\
u_2(k) &= k_{p2}\text{error}_2(k) + k_{d2}\frac{\text{error}_2(k)-\text{error}_2(k-1)}{T} + k_{i2}\sum_{i=1}^{k}\text{error}_2(i)T
\end{aligned}
\tag{6.1}
$$

式中，T 为采样时间，$\text{error}_1(k)=r_1(k)-y_1(k)$，$\text{error}_2(k)=r_2(k)-y_2(k)$。

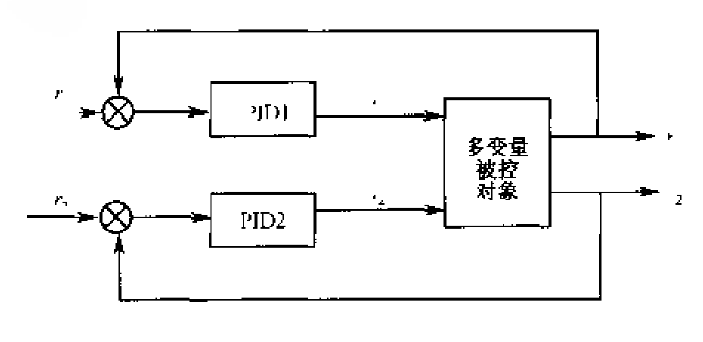


图6-1　二变量PID控制系统框图

6.1.2　仿真程序及分析

仿真实例

设有耦合二变量耦合被控对象：

$$y_1(k)=1.0/(1+y_1(k-1))^2(0.8y_1(k-1)+u_1(k-2)+0.2u_2(k-3))$$

$$y_2(k)=1.0/(1+y_2(k-1))^2(0.9y_2(k-1)+0.3u_1(k-3)+u_2(k-2))$$

设采样时间 $T=1\text{s}$。给定输入为单位阶跃输入，即：

$$\boldsymbol{R}_1=\begin{bmatrix} r_1(k) \\ r_2(k) \end{bmatrix}=\begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\boldsymbol{R}_2=\begin{bmatrix} r_1(k) \\ r_2(k) \end{bmatrix}=\begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

当输入指令为 R_1 时的响应曲线如图 6-2 所示，当输入指令为 R_2 时的响应曲线如图 6-3 所示。

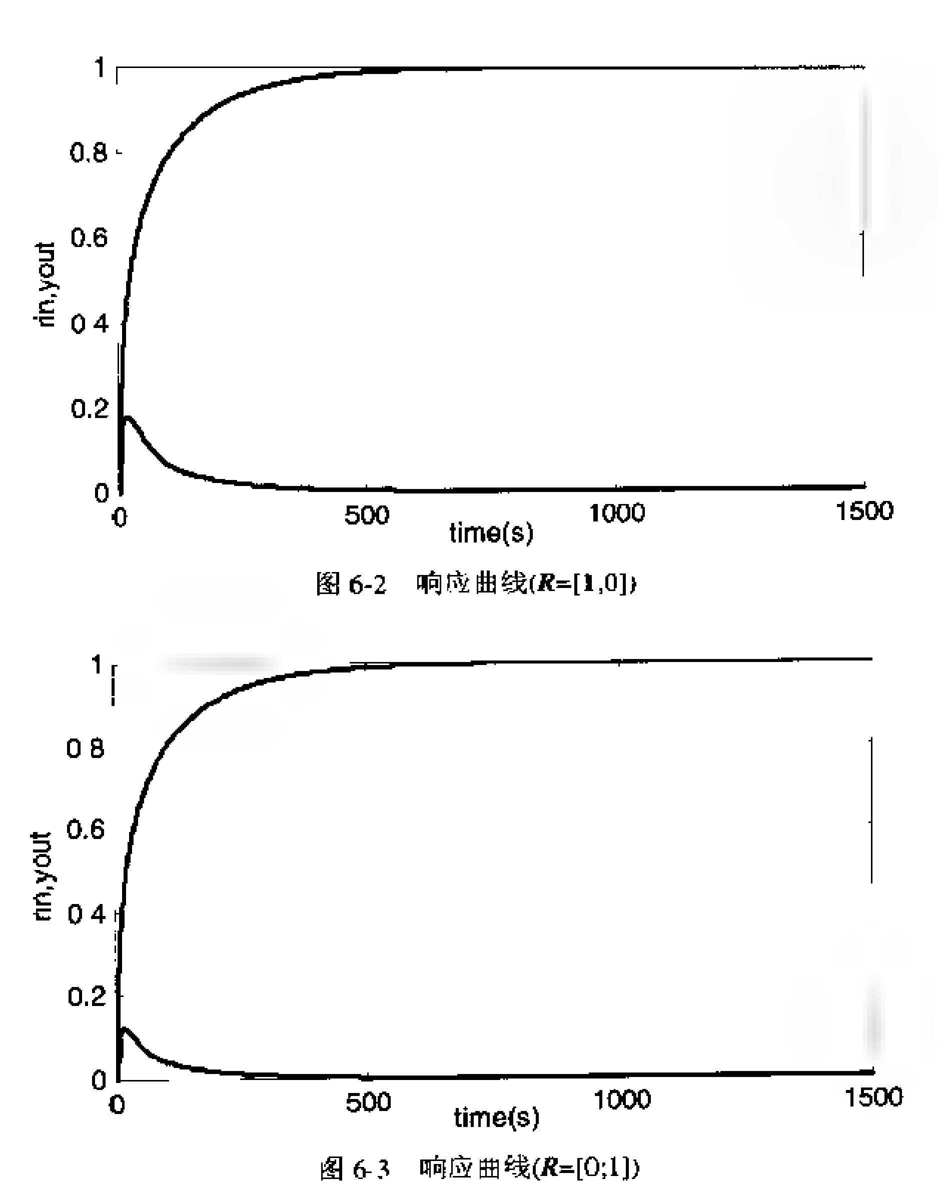


图 6-2　响应曲线(R=[1,0])

图 6-3　响应曲线(R=[0;1])

仿真程序：chap6_1.m。

```
%PID Controller for coupling plant
clear all;
close all;

u1_1=0.0;u1_2=0.0;u1_3=0.0;u1_4=0.0;
u2_1=0.0;u2_2=0.0;u2_3=0.0;u2_4=0.0;

y1_1=0;y2_1=0;

x1=[0;0];x2=[0;0];x3=[0;0];

kp=0.020;
```

```
ki=0.050;
kd=0.0001;

error_1=[0;0];
ts=1;
for k=1:1:1500
time(k)=k*ts;

%Step Signal
%R=[1;0];
R=[0;1];

%PID Decouple Controller
u1(k)=kp*x1(1)+kd*x2(1)+ki*x3(1);
u2(k)=kp*x1(2)+kd*x2(2)+ki*x3(2);
u=[u1(k),u2(k)];

if u1(k)>=10
   u1(k)=10;
end
if u2(k)>=10
   u2(k)=10;
end
if u1(k)<=-10
   u1(k)=-10;
end
if u2(k)<=-10
   u2(k)=-10;
end

%Coupling Plant
yout1(k)=1.0/(1+y1_1)^2*(0.8*y1_1+u1_2+0.2*u2_3);
yout2(k)=1.0/(1+y2_1)^2*(0.9*y2_1+0.3*u1_3+u2_2);

error1(k)=R(1)-yout1(k);
error2(k)=R(2)-yout2(k);
error=[error1(k);error2(k)];

%-------------Return of PID parameters-------------%
u1_4=u1_3;u1_3=u1_2;u1_2=u1_1;u1_1=u(1);
```

6.1.3 多变量 PID 控制的 Simulink 仿真

设有耦合二变量耦合被控对象：

$$y_1(k)=1.0/(1+y_1(k-1)^2)(0.8y_1(k-1)+u_1(k-2)+0.2u_2(k-3))$$

$$y_2(k)=1.0/(1+y_2(k-1)^2)(0.9y_2(k-1)+0.3u_1(k-3)+u_2(k-2))$$

设采样时间 $T=1\text{s}$，给定输入为单位阶跃输入，即：

$$R_1=\begin{bmatrix} r_1(k) \\ r_2(k) \end{bmatrix}=\begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

响应曲线如图 6-4 和图 6-5 所示。

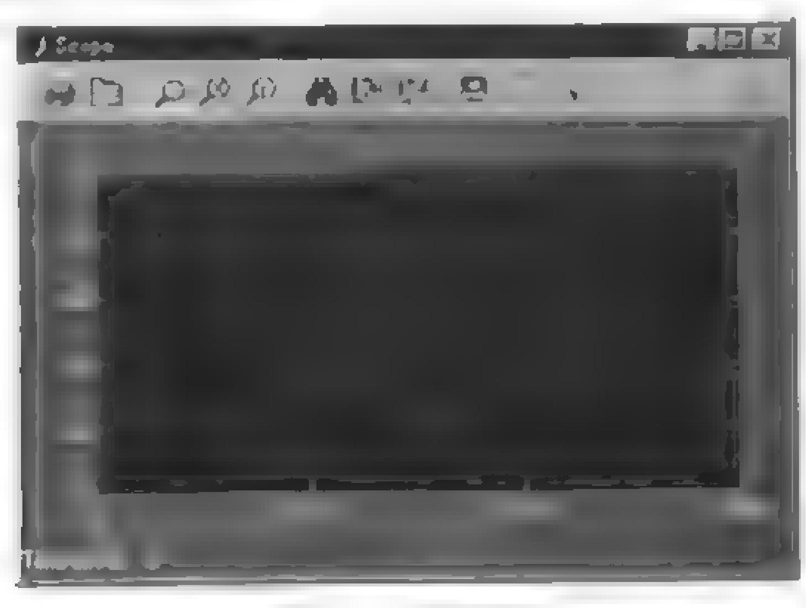


图 6-4 $y_1(k)$ 的响应曲线

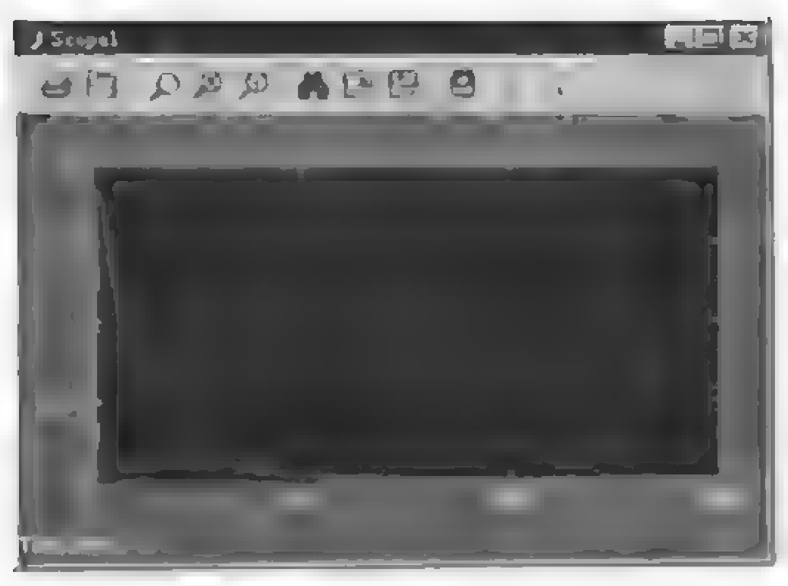


图 6-5 [illegible]

仿真程序的 Simulink 主程序：chap6_2.mdl，如图 6-6 所示。

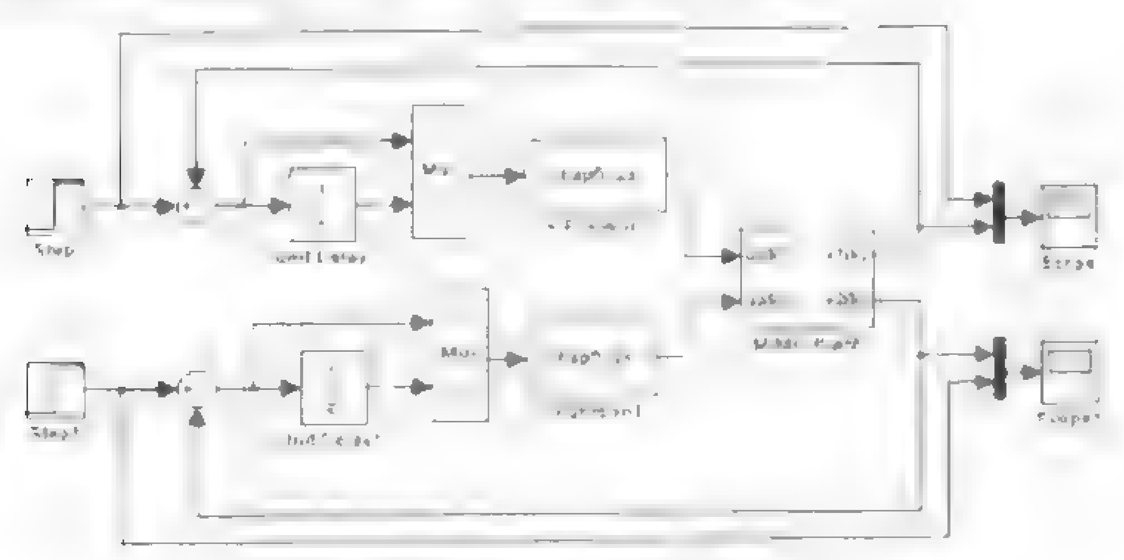


图 6-6 PID 控制的 Simulink 仿真程序

S 函数控制子程序：chap6_2s.m

```
    error (['Unhandled flag = ',num2str(flag)]);
end;

%---------------------------------------------------------------------
% when flag=0, perform system initialization
%---------------------------------------------------------------------
function [sys,x0,str,ts] = mdlInitializeSizes
sizes = simsizes;        % read default control variables
sizes.NumContStates = 0; % no continuous states
sizes.NumDiscStates = 3; % 3 states and assume they are the P/I D components
sizes.NumOutputs = 1;     % 2 output variables: control u(t) and state x(3)
sizes.NumInputs = 2,      % 4 input signals
sizes.DirFeedthrough = 1;% input reflected directly in output
sizes.NumSampleTimes = 1;% single sampling period
sys = simsizes(sizes);   %
x0 = [0; 0; 0];          % zero initial states
str = [];
ts = [1 0],              % sampling period
%---------------------------------------------------------------------
% when flag=2, updates the discrete states
%---------------------------------------------------------------------
function sys = mdlUpdates(x,u
T=1;
sys=[ u(1);
     x(2)+ (1)*T;
     ( 1) u(2 ) T];

%---------------------------------------------------------------------
% when flag=3, computates the output signals
%---------------------------------------------------------------------
function sys = mdlOutputs(t,x,u)

kp=0.02;
ki=0.05;
kd=0.0001;

%sys=[kp,ki,kd]*x;
sys=kp*x(1)+ki*x(2)+kd*x(3);
```

被控对象的 Simulink 仿真子程序：由简单的 Simulink 模块和离散 *S* 函数组成，如图 6-7 所示。

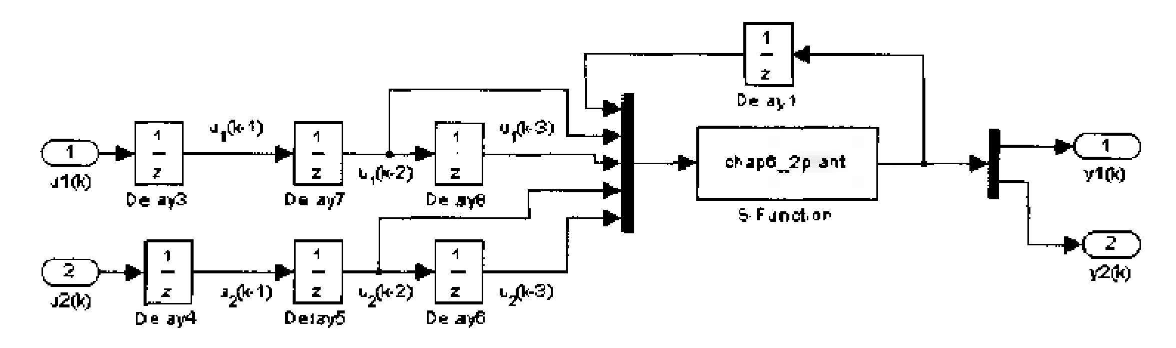


图 6-7　被控对象的 Simulink 仿真子程序

S 函数被控对象子程序：chap6_2plant.m。

```
function [sys,x0,str,ts]=mm_model_ni(t,x,u,flag,T)
switch flag,
case 0 % Initialization
    [sys,x0,str,ts]=mdlInitializeSizes(T);
case 3 % evaluation of outputs
    sys=mdlOutputs(u);
case {1, 2, 4, 9} % undefined flag values
    sys=[];
otherwise % error handling
    error(['Unhandled flag=',num2str(flag)]);
end;

%-----------------------------------------------------------------
% when flag=0, initialization processed for the system
%-----------------------------------------------------------------
function [sys,x0,str,ts]=mdlInitializeSizes(T)
sizes=simsizes; % read the default templates for the system variables
sizes.NumContStates=0; % no continuous states
sizes.NumDiscStates=0; % 6 discrete states, [x1 x3]
sizes.NumOutputs=2; % control variable u(k) and PID parameters
sizes.NumInputs=6; % 7 input signals
sizes.DirFeedthrough=1; % inputs are needed in output evaluation
sizes.NumSampleTimes=1; % single sampling period
sys=simsizes(sizes); % setting of system variables
x0=[]; % zero states, and 0.1 for initial weights
str=[];
ts=[T 0]; % T is the sampling period for the system

function sys=mdlOutputs(u)
sys=[(0.8*u(1)+u(3))+0.2*u(6))/(1+u(1))^2;
     (0.9*u(2)+0.3*u(4)+u(5))/(1+u(2))^2];
```

6.2 单神经元 PID 控制

6.2.1 单神经元 PID 控制原理

通过单神经元 PID 控制，可较好地实现对多变量控制，图 6 8 给出 变量单神经元 PID 控制系统框图，该系统由两个单神经元 PID 控制器构成

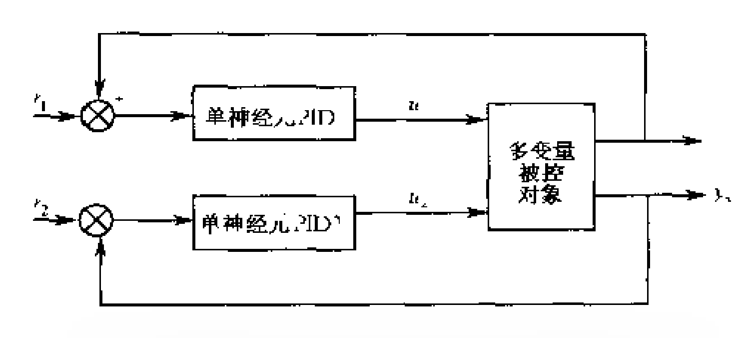


图 6-8 变量单神经元 PID 控制系统框图

单神经元自适应控制器是通过对加权系数的调整来实现自适应、自组织功能，权系数的调整是按有监督的 Hebb 学习规则实现的 以第 个单神经元 PID 控制器为例，控制算法及学习算法为：

$$u_1(k) = u_1(k-1) + k_1\sum_{i=1}^{3} w_i'(k)x_i(k) \tag{6.2}$$

$$\begin{aligned} w_i'(k) &= w_i(k)/\sum_{i=1}^{3}|w_i(k)| \\ w_1(k) &= w_1(k-1) + \eta_I z(k)u(k)x_1(k) \\ w_2(k) &= w_2(k-1) + \eta_P z(k)u(k)x_2(k) \\ w_3(k) &= w_3(k-1) + \eta_D z(k)u(k)x_3(k) \end{aligned} \tag{6.3}$$

式中，$x_1(k) = e(k)$；

$x_2(k) = e(k) - e(k-1)$；

$x_3(k) = \Delta^2 e(k) = e(k) - 2e(k-1) + e(k-2)$，

η_I, η_P, η_D 分别为积分、比例、微分的学习速率，k_1 为神经元的比例系数，$k_1 > 0$。

对积分、比例和微分分别采用了不同的学习速率 η_I, η_P, η_D，以便对不同的权系数分别进行调整。

k_1 值的选择非常重要。k_1 越大，则快速性越好，但超调量大，甚至可能使系统不稳定。当被控对象时延增大时，k_1 值必须减小，以保证系统稳定。k_1 值选择过小，会使系统的快速性变差。

6.2.2 仿真程序及分析

仿真实例

仍以 PID 控制中 变量耦合被控对象为例：

$$y_1(k)=1.0/(1+y_1(k-1))^2(0.8y_1(k-1)+u_1(k-2)+0.2u_2(k-3))$$
$$y_2(k)=1.0/(1+y_2(k-1))^2(0.9y_2(k-1)+0.3u_1(k-3)+u_2(k-2))$$

设采样时间 $T=1\text{s}$。给定输入为单位阶跃输入，即：

$$\boldsymbol{R}_1=\begin{bmatrix} r_1(k) \\ r_2(k) \end{bmatrix}=\begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\boldsymbol{R}_2=\begin{bmatrix} r_1(k) \\ r_2(k) \end{bmatrix}=\begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

采用单神经元 PID 控制，取 $k_1=k_2=0.16$。k_2 为第二个神经元的比例系数。当输入指令为 R_1 时的响应曲线如图 6-9 所示，当输入指令为 R_2 时的响应曲线如图 6-10 所示。

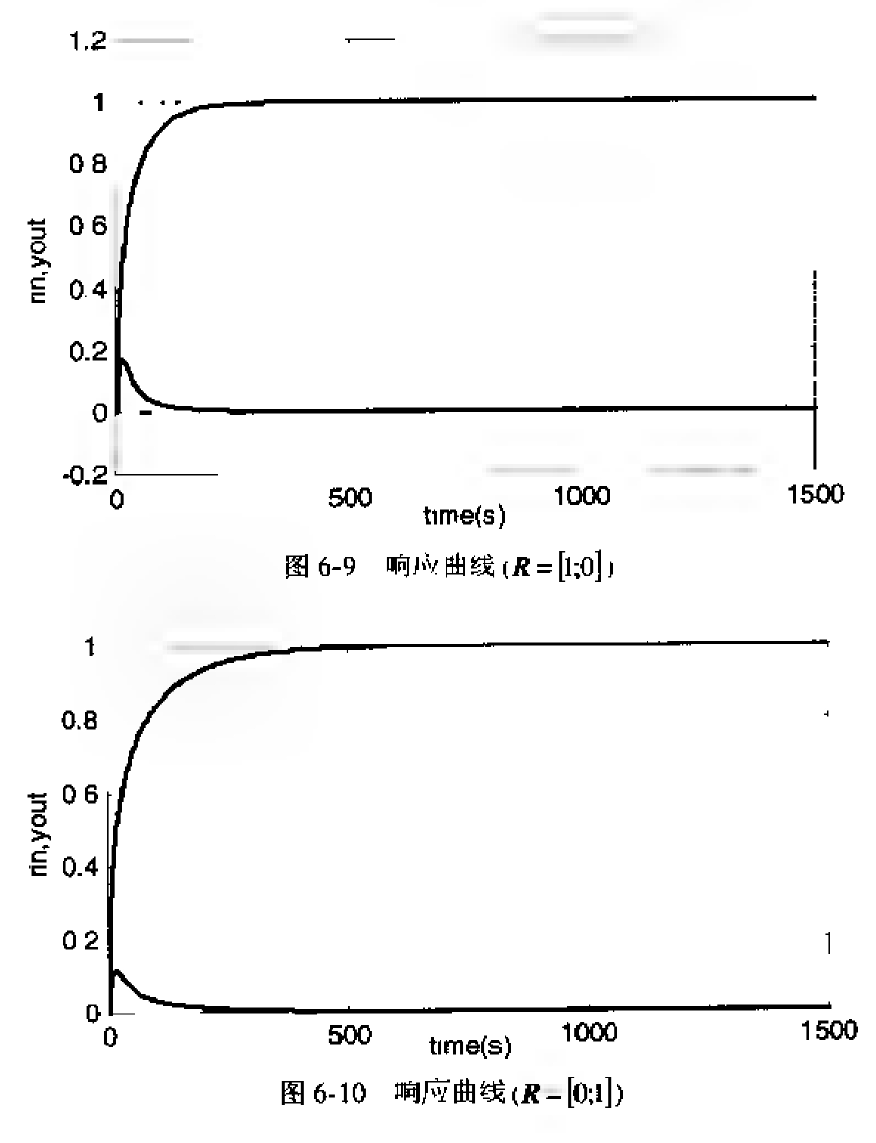


图 6-9　响应曲线（$\boldsymbol{R}=[1;0]$）

图 6-10　响应曲线（$\boldsymbol{R}=[0;1]$）

与 PID 控制相比，单神经元 PID 控制具有响应速度快，自适应能力强，抗干扰能力强等优点。

仿真程序：chap6_3.m。

```
%Single Neural Net PID Decouple Controller based on Hebb Learning
%Algorithm to adjust kp,ki,kd
```

```
clear all;
close all;

xc1=[0,0,0]';
xc2=[0,0,0]';

xiteP=0.40;
xiteI=0.40;
xiteD=0.40;

%Initilizing kp,ki and kd
%Radom Value
%wkp1_1=rand;wki1_1=rand;wkd1_1=rand;
%wkp2_1=rand;wki2_1=rand;wkd2_1=rand;

wkp1_1=0.3150;wki1_1=1.1615;wkd1_1=1.4948;
wkp2_1=0.2067;wki2_1=0.6365;wkd2_1=0.4996;

error1_1=0;error1_2=0;
error2_1=0;error2_2=0;

u1_1=0.0;u1_2=0.0;u1_3=0.0;u1_4=0.0;
u2_1=0.0;u2_2=0.0;u2_3=0.0;u2_4=0.0;

y1_1=0;y2_1=0;

ts=1;
for k=1:1:1500
time(k)=k*ts;

%Step Signal
%R=[1;0];
R=[0;1];

%----------     Calculating practical output     ----------%
%Coupling Plant
yout1(k)=1.0/(1+y1_1)^2*(0.8*y1_1+u1_2+0.2*u2_3);
yout2(k)=1.0/(1+y2_1)^2*(0.9*y2_1+0.3*u1_3+u2_2);

error1(k)=R(1)-yout1(k);
```

```
error2 κ)-R(2) yo_t2,k ;

%For Variable1
%Adjusting NNC Weight Val_e by adopting hebb learning algorithm
  wkp1(k) wkp1 1+xiteP*error1 κ *.1 1*xc1 1 ;  %P
  wk.1(k  wk.1_1+x.teI*error1 κ)*.1 1*xc1 2 ;  %I
  wkd1 k)-wkd1 1+xiteD*error1 k)*u1_1*xc1() ;  %_
  xc1 1) error1(k)-error1 1;              %P
  xc1 2  error1(κ);                      %I
  xc1 3,  error1(κ, 2*error1 1+error1_2 ;      %D

  wadd1 k) abs(wκp1(κ  +abs(wki1 k))+abs(wkd1(κ ,;
  w111(κ  wκp1(κ /wadd1,k);
  w122,k) wki1,k /wadd1(k);
  w133 k,-wkd1 k,/wadd1(k ;
  w1 [w111(k),w122(k ,w133 k)];
  k1-0.16;
  i1(k)  .1_1+k1*w1*xc1,

%For Variable2
%Adjusting NNC Weight Value by adopting hebb learning algorithm
  wkp2,k  wκp2_1+xiteP*error2(k)*.2 1*xc2(1 ;  %P
  wκi2(k -wκi2 1+xiteI*error2 k)*u2_1*xc2(2);  %I
  wκd2 k)-wkd2 1+xiteD*error2 k)*u2_1*xc2(3);  %D
  xc2 1)-error2 k) error2_1;              %P
  xc2(2) error2(k);                      %I
  xc2(3 - error2(κ) 2*error2 1+error2_2);      %D

  wadd2(k  abs(wkp2,k )+abs(wki2 k),+abs(wkd2(k,);
  w211 k -wkp2(k,,wadd2(k),
  w222(k) wki2(k),wadd2(k),
  w233 k) wkd2(k /wadd2(k ;
  w2- w211,k),w222(k),w233(k ],
  k2 0.16;
  u2(k,-.2 1+k2*w1*xc2;

% ------- Ret_rn of PID parameters   --       -%
%For Variable1
error1 2 error1 1;
error1 1 error1 k);
wkp1 1 wkp1,k);
```

```
wkd1_1=wkd1(k);
wki1_1=wki1(k);

u1_4=u1_3;
u1_3=u1_2;
u1_2=u1_1;
u1_1=u1(k);

y1_1=yout1(k);

%For Variable2
error2_2=error2_1;
error2_1=error2(k);
wkp2_1=wkp2(k);
wkd2_1=wkd2(k);
wki2_1=wki2(k);

u2_4=u2_3;
u2_3=u2_2;
u2_2=u2_1;
u2_1=u2(k);

y2_1=yout2(k);
end
figure(1);
plot(time,R(1),'k',time,yout1,'k');
hold on;
plot(time,R(2),'k',time,yout2,'k');
xlabel('time(s)');ylabel('rin,yout');
```

6.2.3 多变量单神经元 PID 控制的 Simulink 仿真

设有耦合二变量耦合被控对象:

$$y_1(k)=1.0/(1+y_1(k-1))^2(0.8y_1(k-1)+u_1(k-2)+0.2u_2(k-3))$$

$$y_2(k)=1.0/(1+y_2(k-1))^2(0.9y_2(k-1)+0.3u_1(k-3)+u_2(k-2))$$

设采样时间$T=1\text{s}$。给定输入为单位阶跃输入，即:

$$\boldsymbol{R}_1=\begin{bmatrix} r_1(k) \\ r_2(k) \end{bmatrix}=\begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

响应曲线如图 6-11 和图 6-12 所示。

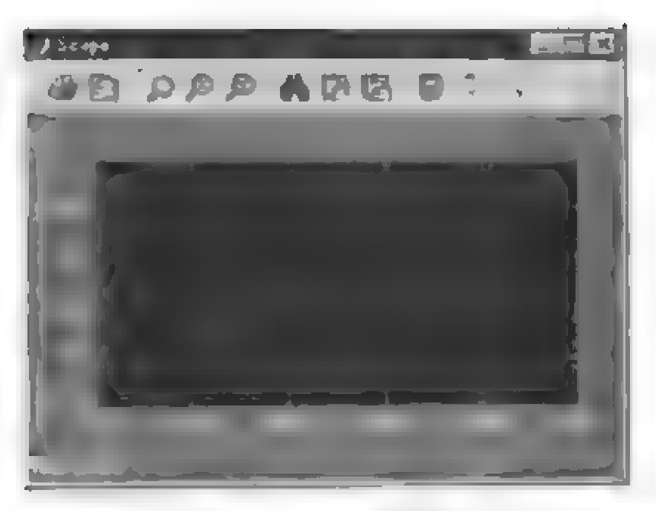


图6-11 [illegible]

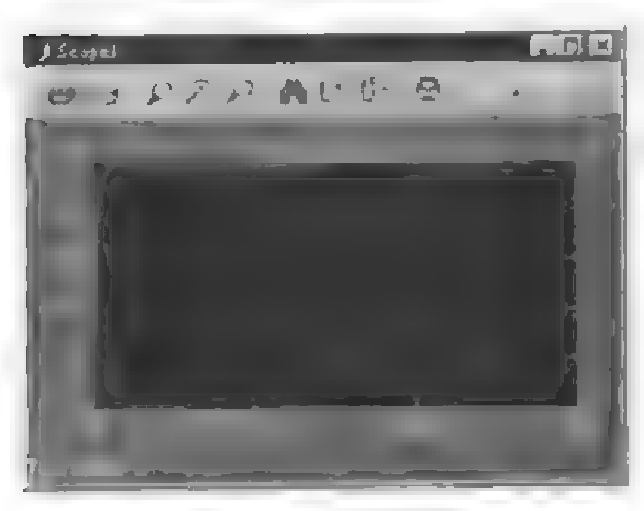


图6-12 [illegible]

仿真程序的Simulink主程序：chap6_4.mdl，如图6-13所示。

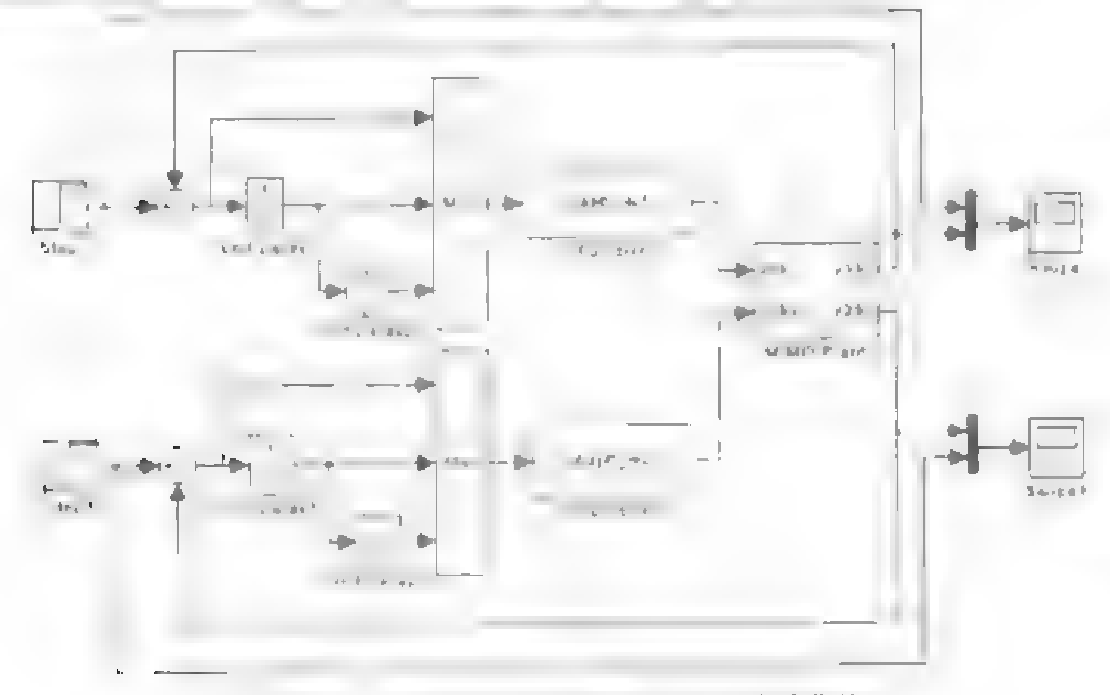


图6-13 单神经元PID控制的Simulink仿真程序

第一个 *S* 函数控制子程序：**chap6_4s1.m**

```
%Single Neural Net PID Decouple Controller based on Hebb Learning
%Algorithm to adjust kp,ki,kd
function [sys,x0,str,ts]=exp_pidf(t,x,u,flag)
switch flag,
case 0          % initializations
   [sys,x0,str,ts] = mdlInitializeSizes;
case 2          % discrete states updates
   sys = mdlUpdates(x,u);
case 3          % computation of control signal
   sys=mdlOutputs(t,x,u);
case {1, 4, 9}   % unused flag values
   sys = [];
otherwise        % error handling
   error(['Unhandled flag = ',num2str(flag)]);
end;

%-------------------------------------------------------------------
% when flag=0, perform system initialization
%-------------------------------------------------------------------
function [sys,x0,str,ts] = mdlInitializeSizes
sizes = simsizes;        % read default control variables
sizes.NumContStates = 0; % no continuous states
sizes.NumDiscStates = 3; % 3 states and assume they are the P/I/D components
sizes.NumOutputs = 1;    % 2 output variables: control u(t) and state x(3)
sizes.NumInputs = 3;     % 4 input signals
sizes.DirFeedthrough = 1;% input reflected directly in output
sizes.NumSampleTimes = 1;% single sampling period
sys = simsizes(sizes);   %
x0 = [0; 0; 0];          % zero initial states
str = [];
ts = [ 1 0];             % sampling period
%-------------------------------------------------------------------
% when flag=2, updates the discrete states
%-------------------------------------------------------------------
function sys = mdlUpdates(x,u)
T=1;
sys=[ u(1);
      x(2)+u(1)*T;
      (u(1)-u(2))/T];

%-------------------------------------------------------------------
```

```
% when flag=3, computates the output signals
% ----------------------------------------------------------------
function sys=mdlOutputs(t,x,u)
persistent wkp1_1 wki1_1 wkd1_1 u1_1
xiteP=0.60;
xiteI=0.60;
xiteD=0.60;

if t==0  %Initilizing kp,ki and kd
   wkp1_1=0.3;
   wki1_1=0.3;
   wkd1_1=0.3;
   u1_1=0;
end

%Adjusting NNC Weight Value by adopting hebb learning algorithm
  wkp1=wkp1_1+xiteP*x(1)*u1_1*x(1);  %P
  wki1=wki1_1+xiteI*x(1)*u1_1*x(2);  %I
  wkd1=wkd1_1+xiteD*x(1)*u1_1*x(3);  %D

  wadd1=abs(wkp1)+abs(wki1)+abs(wkd1);
  w111=wkp1/wadd1;
  w122=wki1/wadd1;
  w133=wkd1/wadd1;
  w1=[w111,w122,w133];
  k1=0.20;
  u1=k1*w1*x;

  wkp1_1=wkp1;
  wkd1_1=wkd1;
  wki1_1=wki1;

  u1_1=u1;

  sys=u1;
```

第二个S函数控制子程序：chap6_4s2.m

```
%Single Neural Net PID Decouple Controller based on Hebb Learning
%Algorithm to adjust kp,ki,kd
function [sys,x0,str,ts]=exp_pidf(t,x,u,flag)
switch flag,
```

```
case 0          % initializations
   [sys,x0,str,ts] = mdlInitializeSizes;
case 2          % discrete states updates
   sys = mdlUpdates(x,u);
case 3          % computation of control signal
   sys=mdlOutputs(t,x,u);
case {1, 4, 9}  % unused flag values
   sys = [];
otherwise       % error handling
   error(['Unhandled flag = ',num2str(flag)]);
end;

%------------------------------------------------------------------
% when flag=0, perform system initialization
%------------------------------------------------------------------
function [sys,x0,str,ts] = mdlInitializeSizes
sizes = simsizes;        % read default control variables
sizes.NumContStates = 0; % no continuous states
sizes.NumDiscStates = 3; % 3 states and assume they are the P/I/D components
sizes.NumOutputs = 1;    % 2 output variables: control u(t) and state x(3)
sizes.NumInputs = 3;     % 4 input signals
sizes.DirFeedthrough = 1;% input reflected directly in output
sizes.NumSampleTimes = 1;% single sampling period
sys = simsizes(sizes);   %
x0 = [0; 0; 0];          % zero initial states
str = [];
ts = [-1 0];             % sampling period
%------------------------------------------------------------------
% when flag=2, updates the discrete states
%------------------------------------------------------------------
function sys = mdlUpdates(x,u)
T=1;
sys=[ u(1);
      x(2)+u(1)*T;
       (u(1)-u(2))/T];

%------------------------------------------------------------------
% when flag=3, computates the output signals
%------------------------------------------------------------------
function sys = mdlOutputs(t,x,u)
persistent wkp2_1 wki2_1 wkd2_1 u2_1
```

```
xiteP=0.60;
xiteI=0.60;
xiteD=0.60;

if t==0 %Initilizing kp,ki and kd
    wkp2_1=0.3;
    wki2_1=0.3;
    wkd2_1=0.3;
    u2_1=0;
end

%Adjusting NNC Weight Value by adopting hebb learning algorithm
   wkp2=wkp2_1+xiteP*x(1)*u2_1*x(1); %P
   wki2=wki2_1+xiteI*x(1)*u2_1*x(2); %I
   wkd2=wkd2_1+xiteD*x(1)*u2_1*x(3); %D

   wadd2=abs(wkp2)+abs(wki2)+abs(wkd2);
   w211=wkp2/wadd2;
   w222=wki2/wadd2;
   w233=wkd2/wadd2;
   w2=[w211,w222,w233];
   k2=0.20;
   u2=k2*w2*x;

   wkp2_1=wkp2;
   wkd2_1=wkd2;
   wki2_1=wki2;

   u2_1=u2;

   sys=u2;
```

被控对象的Simulink子程序及其 S 函数程序同chap6_2.mdl中的被控对象程序。

6.3 基于DRNN神经网络整定的PID控制

6.3.1 基于DRNN神经网络参数自学习PID控制原理

双输入、双输出多变量自整定PID控制器如图6-14所示，其中NN1和NN2为神经网络，用于控制器 u_1 和 u_2 的PID参数为 k_p,k_i,k_d。r_1 和 r_2 为系统输入指令，y_1 和 y_2 为系统输出值

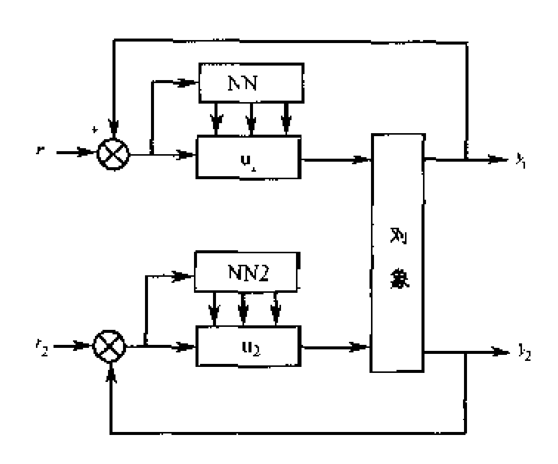


图 6.14 多变量自整定 PID 控制器

以控制器 u_1 为例，控制算法如下：

$$u_1(k)=k_{p1}(k)x_1(k)+k_{i1}(k)x_2(k)+k_{d1}(k)x_3(k) \tag{6.4}$$

$$\text{error}_1(k)=r_1(k)-y_1(k)$$

且有：

$$x_1(k)=\text{error}_1(k)$$

$$x_2(k)=\sum_{i=1}^{k}(\text{error}_1(k)\times T) \tag{6.5}$$

$$x_3(k)=\frac{\text{error}_1(k)-\text{error}_1(k-1)}{T}$$

式中，T 为采样时间。PID 三项系数 $k_{p1}(k)$,$k_{i1}(k)$,$k_{d1}(k)$ 采用 DRNN 神经网络进行整定。

定义如下的指标：

$$E_1(k)=\frac{1}{2}(r_1(k)-y_1(k))^2 \tag{6.6}$$

$$\begin{aligned}k_{p1}(k)&=k_{p1}(k-1)-\eta_p\frac{\partial E_1}{\partial k_{p1}}=k_{p1}(k-1)+\eta_p(r_1(k)-y_1(k))\frac{\partial y_1}{\partial u_1}\frac{\partial u_1}{\partial k_{p1}}\\&=k_{p1}(k-1)+\eta_p(r_1(k)-y_1(k))\frac{\partial y_1}{\partial u_1}x_1(k)\end{aligned} \tag{6.7}$$

$$\begin{aligned}k_{i1}(k)&=k_{i1}(k-1)-\eta_i\frac{\partial E_1}{\partial k_{i1}}=k_{i1}(k-1)+\eta_i(r_1(k)-y_1(k))\frac{\partial y_1}{\partial u_1}\frac{\partial u_1}{\partial k_{i1}}\\&=k_{i1}(k-1)+\eta_i(r_1(k)-y_1(k))\frac{\partial y_1}{\partial u_1}x_2(k)\end{aligned} \tag{6.8}$$

$$\begin{aligned}k_{d1}(k)&=k_{d1}(k-1)-\eta_d\frac{\partial E_1}{\partial k_{d1}}=k_{d1}(k-1)+\eta_d(r_1(k)-y_1(k))\frac{\partial y_1}{\partial u_1}\frac{\partial u_1}{\partial k_{d1}}\\&=k_{d1}(k-1)+\eta_d(r_1(k)-y_1(k))\frac{\partial y_1}{\partial u_1}x_3(k)\end{aligned} \tag{6.9}$$

式中，$\frac{\partial y_1}{\partial u_1}$ 为对象的 Jacobian 信息，该信息可以由 DRNN 网络进行辨识。

6.3.2 DRNN 神经网络的 Jacobian 信息辨识

DRNN（Diagonal Recurrent Neural Network）神经网络是一种回归神经网络，网络结构共有三层，隐层为回归层。DRNN 神经网络的结构如图 6-15 所示。

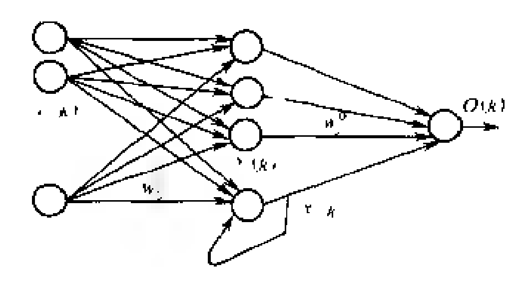


图 6-15 DRNN 神经网络结构

在 DRNN 神经网络中，设 $\boldsymbol{I}=[I_1,I_2,\cdots,I_n]$ 为网络输入向量，$I_i(k)$ 为输入层第 i 个神经元的输入，网络回归层第 j 个神经元的输出为 $X_j(k)$，$S_j(k)$ 为第 j 个回归神经元输入总和，$f(\cdot)$ 为 S 函数，$O(k)$ 为 DRNN 网络的输出。

DRNN 神经网络的算法为：

$$O(k)=\sum_j W_j^{\mathrm{O}}X_j(k),\quad X_j(k)=f(S_j(k)),\quad S_j(k)=W_j^{\mathrm{D}}X_j(k-1)+\sum_i W_{ij}^{\mathrm{I}}I_i(k) \tag{6.10}$$

式中，$\boldsymbol{W}^{\mathrm{D}}$ 和 $\boldsymbol{W}^{\mathrm{O}}$ 为网络回归层和输出层的权值向量，$\boldsymbol{W}^{\mathrm{I}}$ 为网络输入层的权值向量。

在图 6-16 中，k 为网络的迭代步骤，$u(k)$ 和 $y(k)$ 为辨识器的输入。DRNN 为网络辨识器。$y(k)$ 为被控对象实际输出，ym(k) 为 DRNN 的输出。将系统输出 $y(k)$ 及输入 $u(k)$ 的值作为辨识器 DRNN 的输入，将系统输出与网络输出的误差作为辨识器的调整信号。

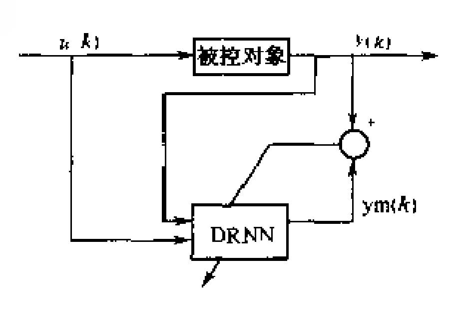


图 6-16 DRNN 神经网络辨识

网络输出层的输出为：

$$\mathrm{ym}(k)=O(k)=\sum_j w_j^{\mathrm{o}}X_j(k) \tag{6.11}$$

网络回归层的输出为：

$$X_j(k)=f(S_j(k)) \tag{6.12}$$

网络回归层的输入为：

$$S_j(k)=w_j^{\mathrm{o}}X_j(k-1)+\sum_i(w_{ij}^{\mathrm{I}}I_i(k)) \tag{6.13}$$

辨识误差为：

$$\mathrm{em}(k)=y(k)-\mathrm{ym}(k) \tag{6.14}$$

辨识指标取：

$$\mathrm{Em}(k)=\frac{1}{2}\mathrm{em}(k)^2 \tag{6.15}$$

学习算法采用梯度下降法：

$$\Delta w_j^{\mathrm{o}}(k)=-\frac{\partial\mathrm{Em}(k)}{\partial w_j^{\mathrm{o}}}=\mathrm{em}(k)\frac{\partial\mathrm{ym}}{\partial w_j^{\mathrm{o}}}=\mathrm{em}(k)X_j(k) \tag{6.16}$$

$$w_j^{\mathrm{o}}(k)=w_j^{\mathrm{o}}(k-1)+\eta_{\mathrm{o}}\Delta w_j^{\mathrm{o}}(k)+\alpha(w_j^{\mathrm{o}}(k-1)-w_j^{\mathrm{o}}(k-2)) \tag{6.17}$$

$$\Delta w_{ij}^{\mathrm{I}}(k)=-\frac{\partial \mathrm{Em}(k)}{\partial w_{ij}^{\mathrm{I}}}=\mathrm{em}(k)\frac{\partial \mathrm{ym}}{\partial w_{ij}^{\mathrm{I}}}=\mathrm{em}(k)\frac{\partial \mathrm{ym}}{\partial X_j}\frac{\partial X_j}{\partial w_{ij}^{\mathrm{I}}}=\mathrm{em}(k)w_j^{\mathrm{o}}Q_{ij}(k) \tag{6.18}$$

$$w_{ij}^{\mathrm{I}}(k)=w_{ij}^{\mathrm{I}}(k-1)+\eta_{\mathrm{I}}\Delta w_{ij}^{\mathrm{I}}(k)+\alpha(w_{ij}^{\mathrm{I}}(k-1)-w_{ij}^{\mathrm{I}}(k-2)) \tag{6.19}$$

$$\Delta w_j^{\mathrm{D}}(k)=-\frac{\partial E(k)}{\partial w_j^{\mathrm{D}}}=\mathrm{em}(k)\frac{\partial \mathrm{ym}}{\partial X_j}\frac{\partial X_j}{\partial w_j^{\mathrm{D}}}=\mathrm{em}(k)w_j^{\mathrm{o}}P_j(k) \tag{6.20}$$

$$w_j^{\mathrm{D}}(k)=w_j^{\mathrm{D}}(k-1)+\eta_{\mathrm{D}}\Delta w_j^{\mathrm{D}}(k)+\alpha(w_j^{\mathrm{D}}(k-1)-w_j^{\mathrm{D}}(k-2)) \tag{6.21}$$

其中，回归层神经元取双 S 函数为：

$$f(x)=\frac{1-\mathrm{e}^{-x}}{1+\mathrm{e}^{-x}} \tag{6.22}$$

$$P_j(k)=\frac{\partial X_j}{\partial w_j^{\mathrm{D}}}=f'(S_j)X_j(k-1) \tag{6.23}$$

$$Q_{ij}(k)=\frac{\partial X_j}{\partial w_{ij}^{\mathrm{I}}}=f'(S_j)I_i(k) \tag{6.24}$$

式中，η_{I}、η_{D}、η_{o} 分别为输入层、回归层和输出层的学习速率，α 为惯性系数。

对象的 Jacobian 信息 $\frac{\partial y}{\partial u}$ 为：

$$\frac{\partial y}{\partial u}\approx\frac{\partial \mathrm{ym}}{\partial u}=\sum_j w_j^{\mathrm{o}}f'(S_j)w_{ij}^{\mathrm{I}} \tag{6.25}$$

6.3.3 仿真程序及分析

仿真实例

仍以 PID 控制中二变量耦合被控对象为例：

$$y_1(k)=1.0/(1+y_1(k-1))^2(0.8y_1(k-1)+u_1(k-2)+0.2u_2(k-3))$$

$$y_2(k)=1.0/(1+y_2(k-1))^2(0.9y_2(k-1)+0.3u_1(k-3)+u_2(k-2))$$

设采样时间为 1s。给定输入为单位阶跃输入，即：

$$\boldsymbol{R}_1=\begin{bmatrix}r_1(k)\\r_2(k)\end{bmatrix}=\begin{bmatrix}1\\0\end{bmatrix}$$

$$\boldsymbol{R}_2=\begin{bmatrix}r_1(k)\\r_2(k)\end{bmatrix}=\begin{bmatrix}0\\1\end{bmatrix}$$

采用基于 DRNN 神经网络整定的 PID 控制，网络结构取 3-7-1，网络输入为 $I=\{u(k-1),y(k),1.0\}$，$\eta_{\mathrm{o}}=0.40$，$\eta_{\mathrm{D}}=0.40$，$\eta_{\mathrm{I}}=0.40$，$\alpha=0.04$，权值取[−1,+1]范围内的随机值。

当输入指令为 $\boldsymbol{R}_1$ 时，Jacobian 信息及响应结果如图 6-17 和图 6-18 所示，$k_{\mathrm{p1}},k_{\mathrm{i1}},k_{\mathrm{d1}}$ 和 $k_{\mathrm{p2}},k_{\mathrm{i2}},k_{\mathrm{d2}}$ 整定结果如图 6-19 和图 6-20 所示。当输入指令为 $\boldsymbol{R}_2$ 时，Jacobian 信息及阶跃响应结果如图 6-21 和图 6-22 所示，$k_{\mathrm{p1}},k_{\mathrm{i1}},k_{\mathrm{d1}}$ 和 $k_{\mathrm{p2}},k_{\mathrm{i2}},k_{\mathrm{d2}}$ 整定结果如图 6-23 和图 6-24 所示。

与 PID 控制相比，基于 DRNN 神经网络整定的 PID 多变量控制具有响应速度快，自适

应能力强，抗干扰能力强等优点。

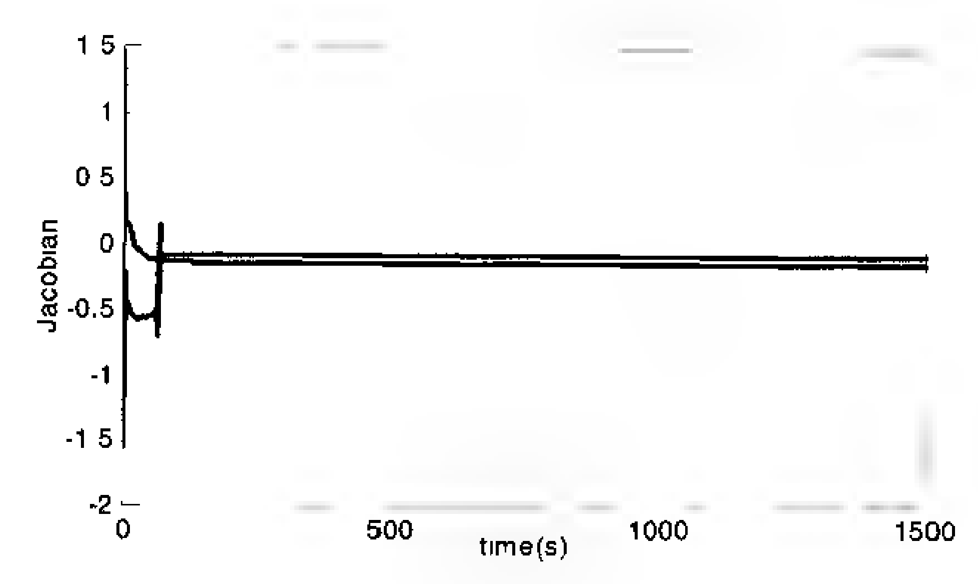


图 6-17　Jacobian 信息($\boldsymbol{R}=[1;0]$)

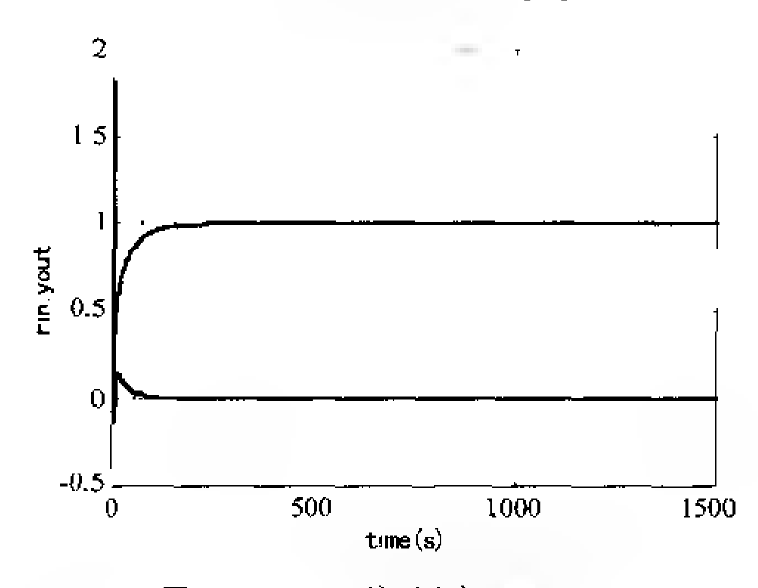


图 6-18　DRNN 控制响应($\boldsymbol{R}=[1;0]$)

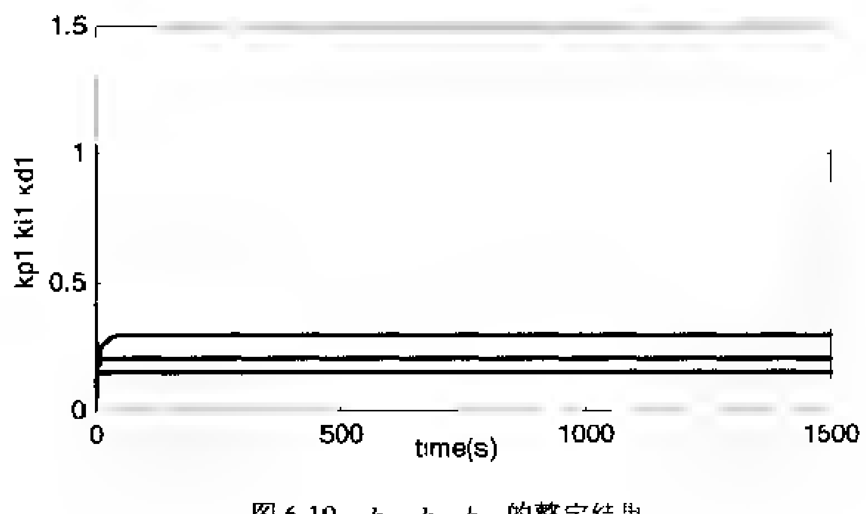


图 6-19　k_{p1}、k_{i1}、k_{d1} 的整定结果

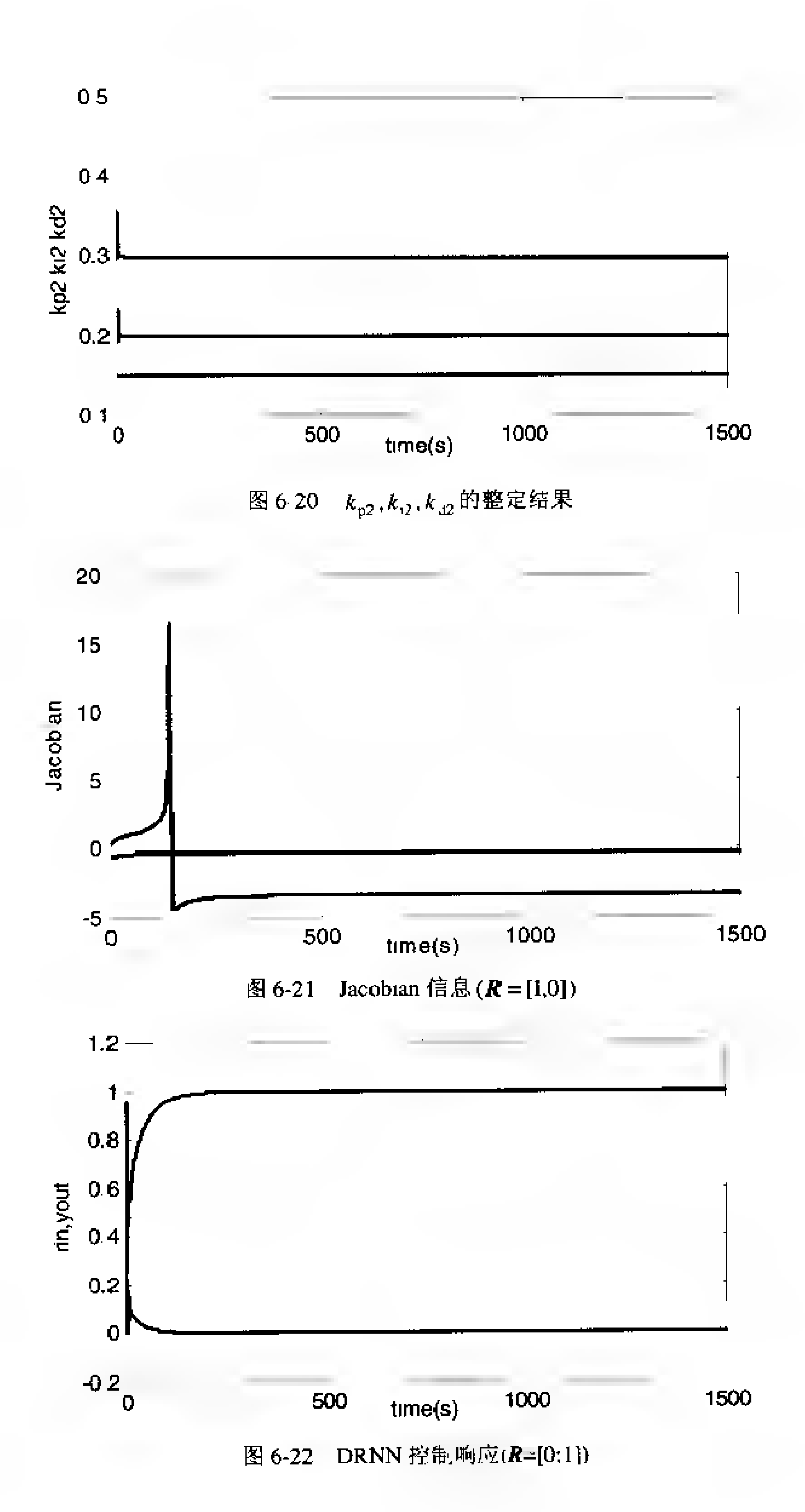


图 6-20　k_{p2}, k_{i2}, k_{d2} 的整定结果

图 6-21　Jacobian 信息 ($\boldsymbol{R}=[1,0]$)

图 6-22　DRNN 控制响应($\boldsymbol{R}=[0;1]$)

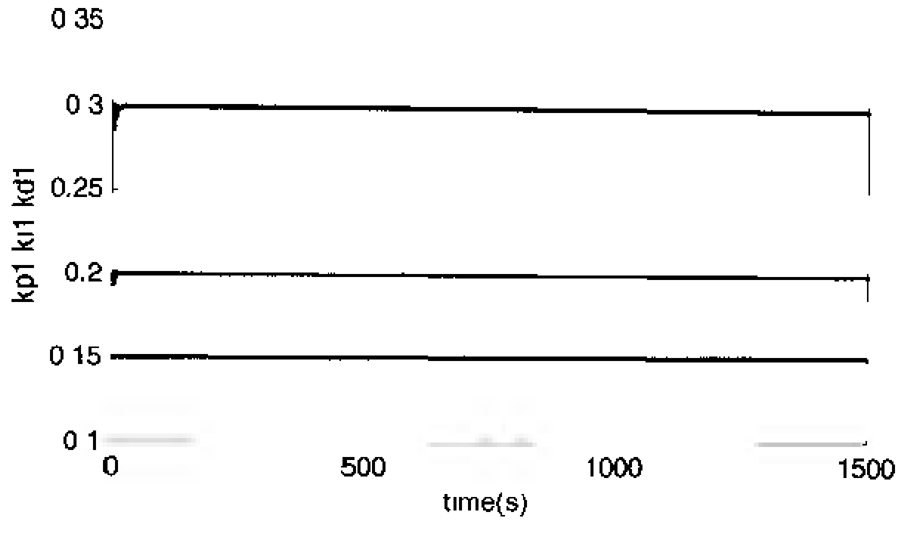


图 6-23 k_{p1}, k_{i1}, k_{d1} 的整定结果

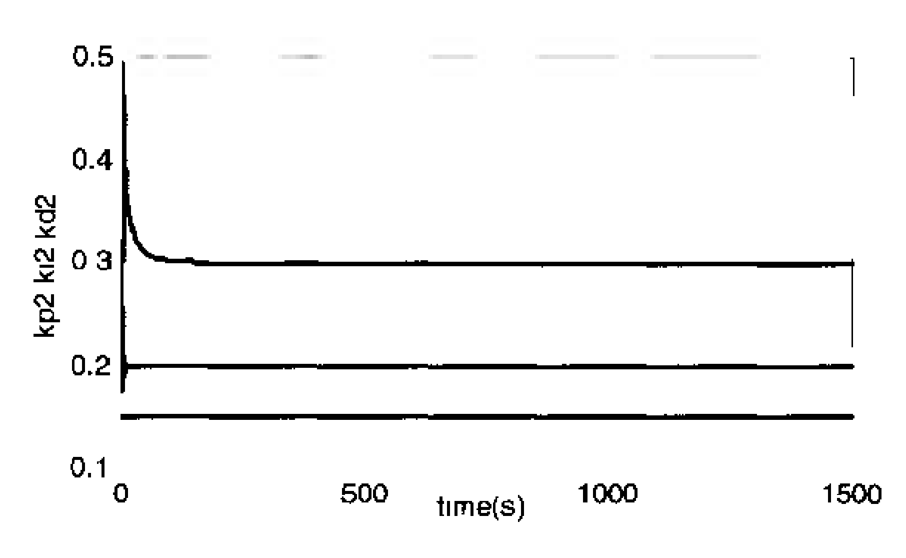


图 6-24 k_{p2}, k_{i2}, k_{d2} 的整定结果

仿真程序：chap6 5.m。

```
%DRNN Tunning PID Controller for coupling plant
clear all;
close all;

u1_1=0.0;u1_2=0.0;u1_3=0.0;
u2_1=0.0;u2_2=0.0;u2_3=0.0;
y1_1=0;y2_1=0;

wd1=rands(7,1);
wo1=rands(7,1);
wi1=rands(3,7);
wd1_1=wd1;wo1_1=wo1;wi1_1=wi1;
```

```
x1=zeros(7,1);
x1_1=x1;

wd2=rands(7,1);
wo2=rands(7,1);
wi2=rands(3,7);
wd2_1=wd2;wo2_1=wo2;wi2_1=wi2;

xitei1=0.40;xited1=0.40;xiteo1=0.40;
xitei2=0.40;xited2=0.40;xiteo2=0.40;
alfa1=0.04;alfa2=0.04;

x2=zeros(7,1);
x2_1=x2;

error1_1=0;error1_2=0;
error2_1=0;error2_2=0;

kp1=0.30;ki1=0.150;kd1=0.2;
kp2=0.30;ki2=0.150;kd2=0.2;

kp1_1=kp1;kd1_1=kd1;ki1_1=ki1;
kp2_1=kp2;kd2_1=kd2;ki2_1=ki2;

xitekp1=0.5;xitekd1=0.3;xiteki1=0.001;
xitekp2=0.5;xitekd2=0.3;xiteki2=0.0001;

ei1=0;ei2=0;
ts=1;
for k=1:1:1500
time(k)=k*ts;

%Step Signal
R=[1;0];
%R=[0;1];

%Coupling Plant
yout1(k)=1.0/(1+y1_1)^2*(0.8*y1_1+u1_2+0.2*u2_3);
yout2(k)=1.0/(1+y2_1)^2*(0.9*y2_1+0.3*u1_3+u2_2);

In1=[u1_1,yout1(k),1]';
for j=1:1:7
```

```
    s1(j)=In1'*wi1(:,j)+wd1(j)*x1(j);
end

for j=1:1:7
    x1(j)=(1-exp(-s1(j)))/(1+exp(-s1(j)));
end

P1=0*x1;
for j=1:1:7
    P1(j)=wo1(j)*(1+x1(j))*(1-x1(j))*x1_1(j);
end

Q1=0*wi1;
for j=1:1:7
   for i=1:1:3
      Q1(i,j)=wo1(j)*(1+x1(j))*(1-x1(j))*In1(i);
   end
end

ym=0;
for j=1:1:7
    ym=ym+x1(j)*wo1(j);
end
ymout1(k)=ym;

wo1=wo1+xiteo1*(yout1(k)-ymout1(k))*x1+alfa1*(wo1-wo1_1);
wd1=wd1+xited1*(yout1(k)-ymout1(k))*P1+alfa1*(wd1-wd1_1);
for j=1:1:7
   if abs(wd1(j))>1
      wd1(j)=0.5*sign(wd1(j));
   end
end
wi1=wi1+xitei1*(yout1(k)-ymout1(k))*Q1+alfa1*(wi1-wi1_1);

yu=0;
for j=1:1:7
   yu=yu+wo1(j)*wi1(1,j)*(1+x1(j))*(1-x1(j));
end
dyout1(k)=yu;

error1(k)=R(1)-yout1(k);
```

```
xc1(1)=error1(k);             %Calculating P
xc1(2)=error1(k)-error1_1;    %Calculating D
ei1=ei1+error1(k)*ts;
xc1(3)=ei1;                   %Calculating I

kp1(k)=kp1_1+xitekp1*error1(k)*xc1(1);
kd1(k)=kd1_1+xitekd1*error1(k)*xc1(2);
ki1(k)=ki1_1+xiteki1*error1(k)*xc1(3);

kp1(k)=kp1_1+xitekp1*error1(k)*xc1(1)*dyout1(k);
kd1(k)=kd1_1+xitekd1*error1(k)*xc1(2)*dyout1(k);
ki1(k)=ki1_1+xiteki1*error1(k)*xc1(3)*dyout1(k);

if kp1(k)<0
   kp1(k)=0;
end
if kd1(k)<0
   kd1(k)=0;
end
if ki1(k)<0
   ki1(k)=0;
end

u1(k)=kp1(k)*xc1(1)+kd1(k)*xc1(2)+ki1(k)*xc1(3);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
In2=[u2_1,yout2(k),1]';

for j=1:1:7
   si(j)=In1'*wi2(:,j)+wd2(j)*x2(j);
end

for j=1:1:7
   x2(j)=(1-exp(-si(j)))/(1+exp(-si(j)));
end

Pi=0*x2;
for j=1:1:7
   Pi(j)=wo2(j)*(1+x2(j))*(1-x2(j))*x2_1(j);
end

Qi=0*wi2;
```

```
for j=1:1:7
   for i=1:1:3
      Qi(i,j)=wo2(j)*(1+x2(j))*(1-x2(j))*In2(i);
   end
end

ym=0;
for j=1:1:7
   ym=ym+x2(j)*wo2(j);
end
ymout2(k)=ym;

wo2=wo2+xiteo2*(yout2(k)-ymout2(k))*x2+alfa2*(wo2-wo2_1);
wd2=wd2+xited2*(yout2(k)-ymout2(k))*Pi+alfa2*(wd2-wd2_1);
  for j=1:1:7
     if abs(wd2(j))>1
        wd2(j)=0.5*sign(wd2(j));
     end
  end
wi2=wi2+xitei2*(yout2(k)-ymout2(k))*Qi+alfa2*(wi2-wi2_1);

yu=0;
for j=1:1:7
    yu=yu+wo2(j)*wi2(1,j)*(1+x2(j))*(1-x2(j));
end
dyout2(k)=yu;

error2(k)=R(2)-yout2(k);

xc2(1)=error2(k);              %Calculating P
xc2(2)=error2(k)-error2_1;     %Calculating D
ei2=ei2+error2(k)*ts;
xc2(3)=ei2;                    %Calculating I

kp2(k)=kp2_1+xitekp2*error2(k)*xc2(1)*dyout2(k);
kd2(k)=kd2_1+xitekd2*error2(k)*xc2(2)*dyout2(k);
ki2(k)=ki2_1+xiteki2*error2(k)*xc2(3)*dyout2(k);

if kp2(k)<0
   kp2(k)=0;
end
if kd2(k)<0
```

```
    kd2(k)=0;
end
if ki2(k)<0
    ki2(k)=0;
end

u2(k)=kp2(k)*xc2(1)+kd2(k)*xc2(2)+ki2(k)*xc2(3);

%Return of PID parameters
error1_2=error1_1;
error1_1=error1(k);

error2_2=error2_1;
error2_1=error2(k);

wd1_1=wd1;wo1_1=wo1;wi1_1=wi1;
wd2_1=wd2;wo2_1=wo2;wi2_1=wi2;

u1_4=u1_3;u1_3=u1_2;u1_2=u1_1;u1_1=u1(k);
u2_4=u2_3;u2_3=u2_2;u2_2=u2_1;u2_1=u2(k);

y1_1=yout1(k);
y2_1=yout2(k);
end
figure(1);
plot(time,dyout1,'r');
xlabel('time(s)');ylabel('Jacobian');
hold on;
plot(time,dyout2,'b');
xlabel('time(s)');ylabel('Jacobian');
figure(2);
plot(time,R(1),'b',time,yout1,'r');
hold on;
plot(time,R(2),'r',time,yout2,'b');
xlabel('time(s)');ylabel('rin,yout');
figure(3);
plot(time,kp1,'r',time,ki1,'b',time,kd1,'k');
xlabel('time(s)');ylabel('kp1 ki1 kd1');
figure(4);
plot(time,kp2,'r',time,ki2,'b',time,kd2,'k');
xlabel('time(s)');ylabel('kp2 ki2 kd2');
```

第 7 章　几种先进 PID 控制方法

7.1　基于干扰观测器的 PID 控制

7.1.1　干扰观测器设计原理

干扰观测器的基本思想是，将外部力矩干扰及模型参数变化造成的实际对象与名义模型输出的差异等效到控制输入端，即观测出等效干扰，在控制中引入等效的补偿，实现对干扰完全抑制[17,18]。其基本结构如图 7-1 所示。

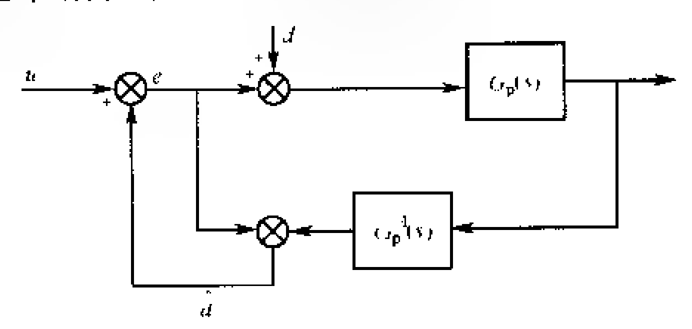


图 7-1　干扰观测器的基本结构

图中的 $G_p(s)$ 为对象的传递函数，d 为等效干扰，$\hat{d}$ 为观测的干扰，u 为控制输入。由此图可求出等效干扰的估计值 $\hat{d}$ 为：

$$\hat{d}=(e+d)\cdot G_p(s)\cdot G_p^{-1}(s)-e=d \tag{7.1}$$

对于实际的物理系统，其实现存在如下问题：

（1）在通常情况下，$G_p(s)$ 的相对阶不为零，其逆在物理上不可实现；

（2）对象 $G_p(s)$ 的精确数学模型无法得到；

（3）考虑到测量噪声的影响，该方法的控制性能将下降。

解决上述问题的惟一方法是在 $\hat{d}$ 的后面串入低通滤波器 $Q(s)$，并用名义模型 $G_n(s)$ 的逆 $G_n^{-1}(s)$ 来代替 $G_p(s)$，从而得到如图 7-2 所示的干扰观测器原理框图，其中虚线部分为干扰观测器。

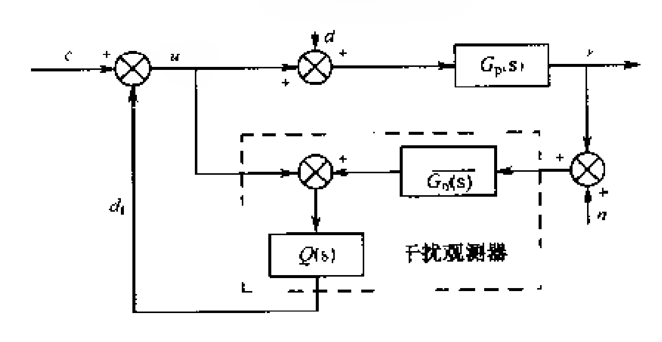


图 7-2　干扰观测器原理框图

控制器的输出为：

$$u = c - \tilde{d}_f + d \tag{7.2}$$

式中，c 为 PID 控制器的输出，$\tilde{d}_f$ 为干扰 d 的估计值。

由图 7-2 可得：

$$G_{CY}(s) = \frac{Y(s)}{C(s)} = \frac{G_p(s)G_n(s)}{G_n(s)+Q(s)(G_p(s)-G_n(s))} \tag{7.3}$$

$$G_{DY}(s) = \frac{Y(s)}{D(s)} = \frac{G_p(s)G_n(s)(1-Q(s))}{G_n(s)+Q(s)(G_p(s)-G_n(s))} \tag{7.4}$$

$$G_{NY}(s) = \frac{Y(s)}{N(s)} = \frac{G_p(s)Q(s)}{G_n(s)+Q(s)(G_p(s)-G_n(s))} \tag{7.5}$$

设低通滤波器 $Q(s)$ 的频带为 f_q。当 $f<f_q$ 时，$Q\approx 1$，$G_{CY}\approx G_n(s), G_{DY}\approx 0, G_{NY}\approx 1$。当 $f>f_q$ 时，$Q\approx 0$，$G_{CY}(s)\approx G_p(s)$，$G_{DY}(s)\approx G_p(s)$，$G_{NY}(s)\approx 0$。通过低通滤波器 $Q(s)$ 的设计可较好地抵抗外加干扰。

由上面分析可见，$Q(s)$ 的设计是干扰观测器设计中的一个重要环节。首先，为使 $Q(s)G_n^{-1}(s)$ 正则，$Q(s)$ 的相对阶应不小于 $G_n(s)$ 的相对阶；其次，$Q(s)$ 带宽的设计应是在干扰观测器的鲁棒稳定性和干扰抑制能力之间的折中。

设 $G_p(s)$ 的名义模型为 $G_n(s)$，则不确定对象的集合可以用乘积摄动来描述，即：

$$G_p(s) = G_n(s)(1+\Delta(s)) \tag{7.6}$$

式中，$\Delta(s)$ 为可变的传递函数。

图 7-3 示出转台伺服系统某框的实测频率特性 $G_p(s)$ 与名义模型 $G_n(s)$ 频率特性，由图可见，当频率增加时，对象的不确定性增大，$|\Delta(jw)|$ 表现为频率 ω 的增函数。

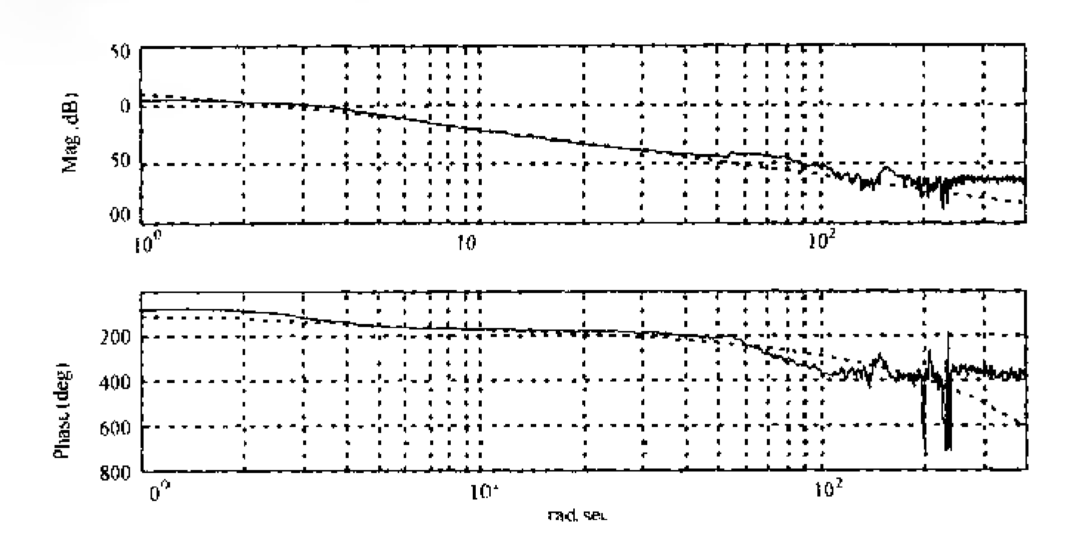


图 7-3　实测被控对象 $G_p(s)$ 与名义模型 $G_n(s)$ 频率特性

由鲁棒稳定性定理，干扰观测器 $Q(s)$ 鲁棒稳定的充分条件：

$$\|\Delta(s)Q(s)\|_\infty < 1 \tag{7.7}$$

式（7.7）是$Q(s)$设计的基础，通过$Q(s)$的设计，可实现鲁棒性要求。

忽略非建模动态和不确定性的影响，$G_n(s)$可描述为：

$$G_n(s)=\frac{1}{s(J_n s+b_n)} \tag{7.8}$$

式中，J_n为等效惯性力矩，b_n为等效阻尼系数。

采用如下形式的低通滤波器：

$$Q(s)=\frac{3\tau s+1}{\tau^3 s^3+3\tau^2 s^2+3\tau s+1} \tag{7.9}$$

由$\Delta(s)=G_p(s)/G_n(s)-1$可得$\Delta(s)$的频率特性，它表明了实际对象频率特性对名义模型的摄动，$\Delta(s)$和不同带宽的$Q(s)$的幅频特性如图7-4所示。可见当$Q(s)=Q_2(s)$时鲁棒稳定性可以得到满足，并且外界干扰可以得到很好的抑制。因此$Q(s)=Q_2(s)$为理想的低通滤波器，此时$\tau=0.001$。

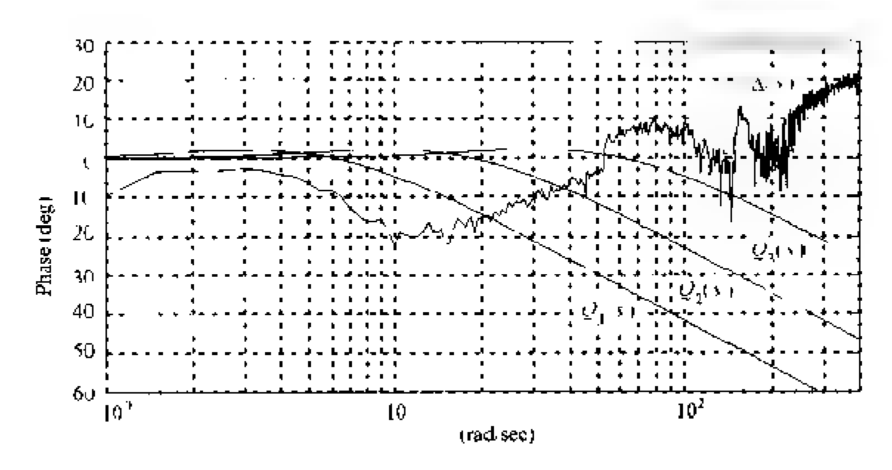


图7-4　$\Delta(s)$和不同带宽的$Q(s)$的幅频特性

7.1.2　连续系统的控制仿真

仿真实例

设实际的被控对象为：

$$G_p(s)=\frac{1}{0.003s^2+0.067s}$$

名义模型取：

$$G_n(s)=\frac{1}{0.0033s^2+0.0673s}$$

取指令信号为：$r(t)=1.0\sin(2\pi t)$，干扰信号为：$d(t)=3\sin(5\pi t)$，PID控制器中取$k_p=5.0, k_i=0, k_d=0.50$。$Q(s)$按式（7.9）进行设计，并取$\tau=0.001$。干扰观测器的Simulink仿真程序如图7-5所示，先运行参数初始化程序chap7_1f.m，对干扰信号的观测结果如图7-6所示，分别对加入干扰观测器和不加入干扰观测器两种情况进行仿真，其正弦跟踪如图7-7和图7-8所示。

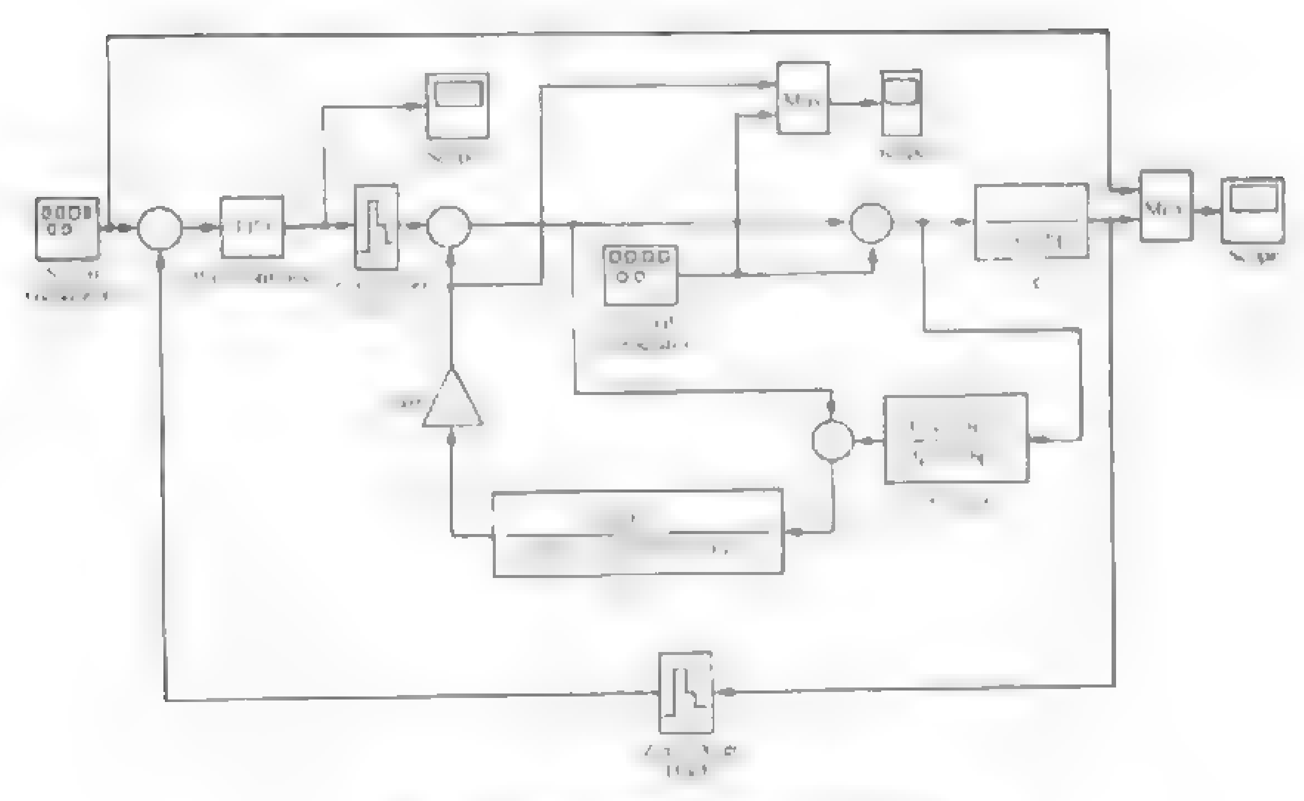


图7-5 干扰观测器的 Simulink 仿真程序

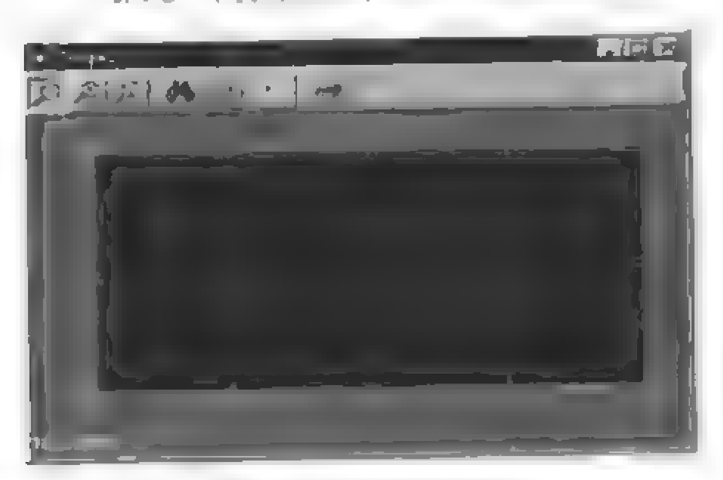


图7-6 [illegible]

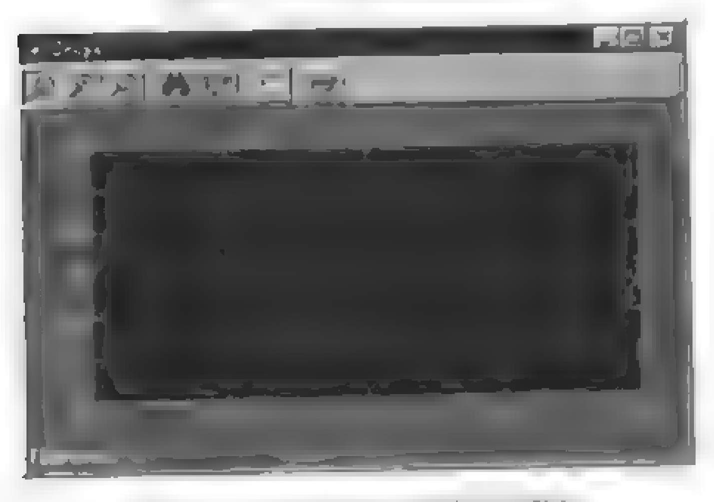


图7-7 [illegible]

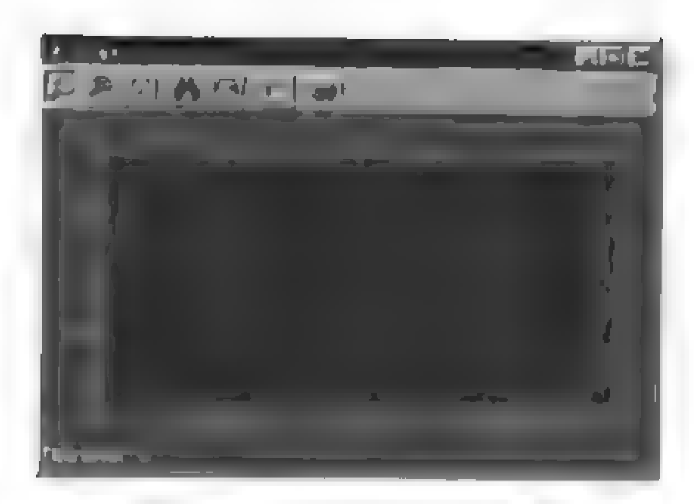


图7-8 有干扰观测器时的正弦跟踪

参数初始化程序：chap7_1f.m

```
[illegible]
clear all
[illegible]
[illegible]
p=tf([1],[Jp,bp,0]);
[illegible]

[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```

仿真主程序：chap7_1.mdl，如图7-5所示。

7.1.3 离散系统的控制仿真

由连续干扰观测器可得到离散干扰观测器的结构，如图7-9所示。$Q(z^{-1})$为低通滤波器，则有：

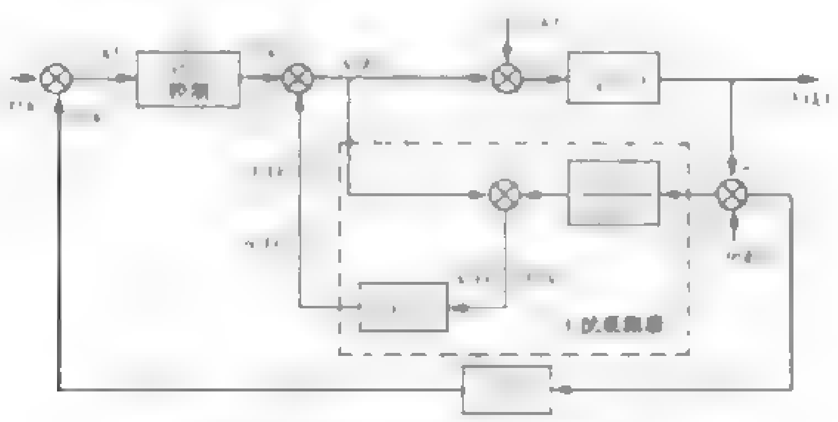


图7-9 离散干扰观测器的结构

$$G_{CY}(z^{-1})=\frac{G_p(z^{-1})G_n(z^{-1})}{G_n(z^{-1})+Q(z^{-1})(G_p(z^{-1})-G_n(z^{-1}))} \tag{7.10}$$

$$G_{DY}(z^{-1})=\frac{G_p(z^{-1})G_n(z^{-1})(1-Q(z^{-1}))}{G_n(z^{-1})+Q(z^{-1})(G_p(z^{-1})-G_n(z^{-1}))} \tag{7.11}$$

$$G_{NY}(z^{-1})=\frac{G_p(z^{-1})Q(z^{-1})}{G_n(z^{-1})+Q(z^{-1})(G_p(z^{-1})-G_n(z^{-1}))} \tag{7.12}$$

设$Q(z^{-1})$为理想的低通滤波器，即在低频段，当$f \leqslant f_q$时，$Q(z^{-1})\approx 1$；在高频段，当$f \geqslant f_q$时，$Q(z^{-1})\approx 0$。

在低频段时，有$G_{CY}(z^{-1})\approx G_n(z^{-1}), G_{DY}(z^{-1})\approx 0, G_{NY}(z^{-1})\approx 1$，说明干扰观测器对于低频干扰具有很好的抑制能力，但对低频测量噪声非常敏感。在高频段时，有$G_{CY}(z^{-1})\approx G_p(z^{-1}), G_{DY}(z^{-1})\approx G_p(z^{-1}), G_{NY}(z^{-1})\approx 0$，说明干扰观测器对于高频段测量噪声具有很好的抑制能力，但对干扰却没有抑制作用。

正确选择$Q(z^{-1})$可实现对干扰$d(k)$和测量噪声$n(k)$的完全抑制。

仿真实例

设实际的被控对象为：

$$G_p(s)=\frac{1}{0.003s^2+0.067s}$$

名义模型取：

$$G_n(s)=\frac{1}{0.00305s^2+0.0671s}$$

采样时间为0.001s。假设干扰信号为$d(k)=50\sin(10\pi t)$，$n(k)$为幅值0.001的随机信号，指令信号为正弦信号：$r(k)=0.50\sin(6\pi t)$，在PD控制中选取$k_p=15.0, k_d=5.0$。

$Q(s)$按式（7.9）进行设计，并取$\tau=0.001$。图7-10为$Q(s)$滤波前、后信号，图7-11为干扰d及其干扰观测器的观测结果$\tilde{d}_f$，图7-12为不加干扰观测器时的正弦跟踪($M=1$)，图7-13为加入干扰观测器时的正弦跟踪($M=2$)。

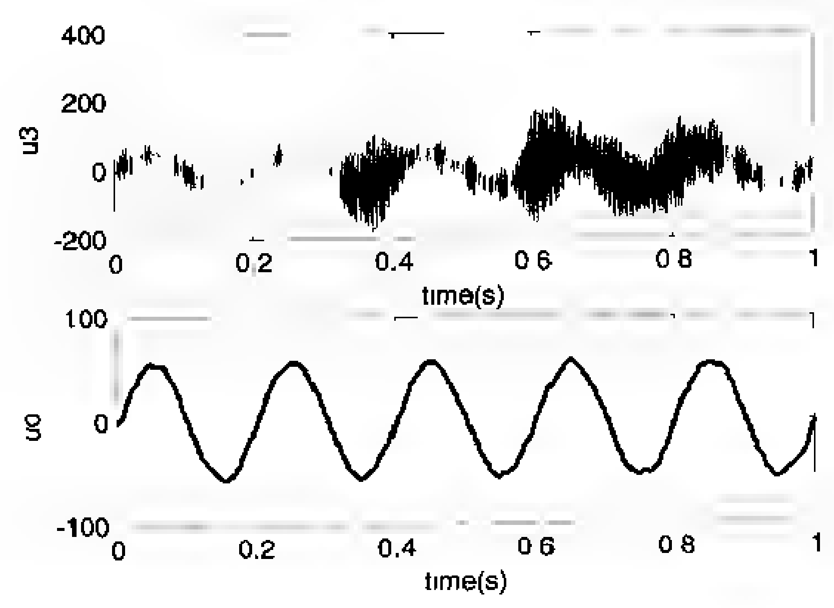


图7-10　低通滤波器滤波前、后信号

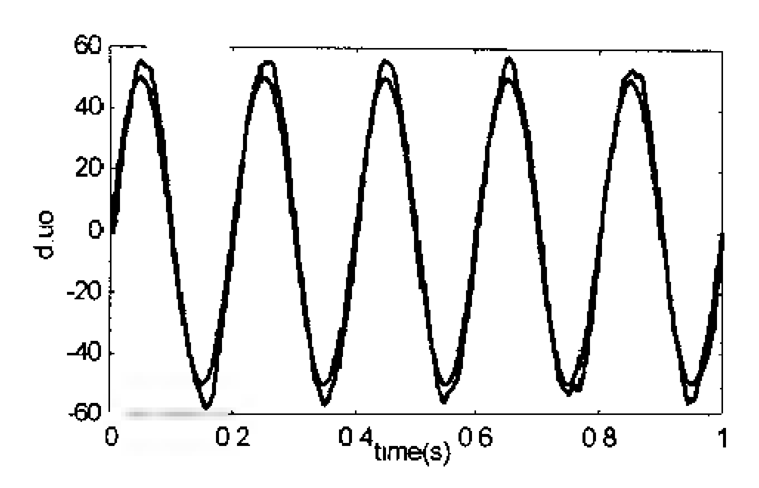


图 7-11 干扰 d 及其干扰观测器的观测结果 $\tilde{d}_f$

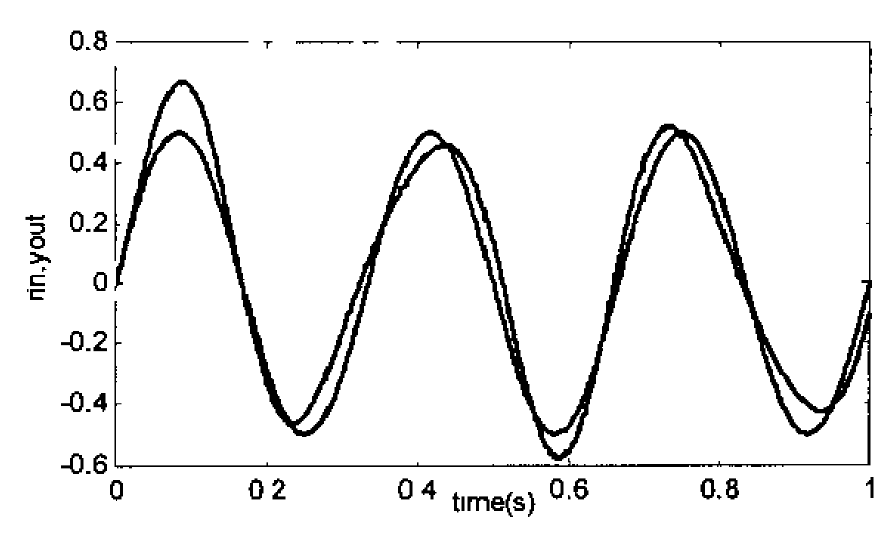


图 7-12 无干扰观测器时的正弦跟踪（M=1）

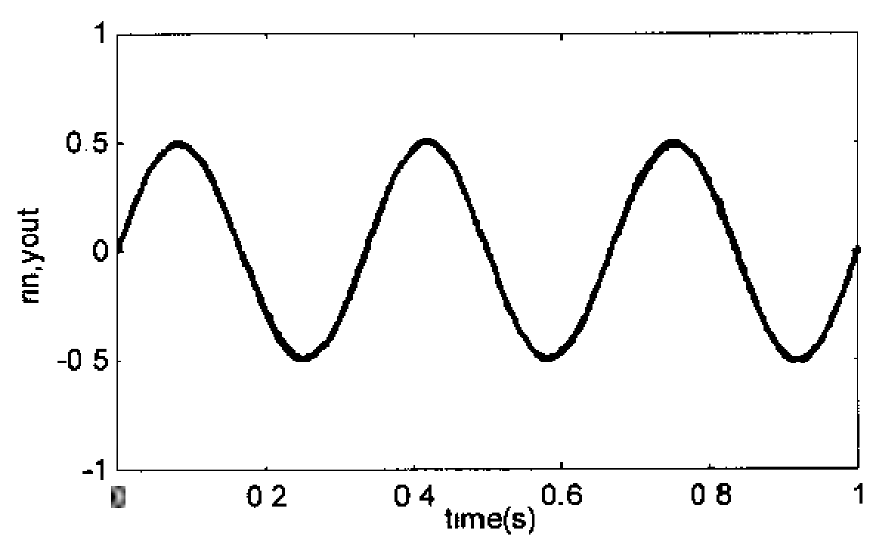


图 7-13 加入干扰观测器时的正弦跟踪（M=2）

仿真程序：chap7_2.m。

```
%PID Control based on Disturbance Observer
clear all;
```

```
close all;

Jp=0.0075;bp=0.1880;
Jn=Jp;bn=bp;

ts=0.001;

Gp=tf([1],[Jp,bp,0]);   %Plant
Gpz=c2d(Gp,ts,'z');
[num,den]=tfdata(Gpz,'v');

Gn=tf([1],[Jn,bn,0]);   %Nominal model
Gnz=c2d(Gn,ts,'z');
[num1,den1]=tfdata(Gnz,'v');

tol=0.0065;
Q=tf([3*tol,1],[tol^3,3*tol^2,3*tol,1]);   %Low Pass Filter
Qz=c2d(Q,ts,'tustin');
[numq,denq]=tfdata(Qz,'v');

uu_1=0;
u2_1=0;
uo_1=0;uo_2=0;uo_3=0;
u3_1=0;u3_2=0;u3_3=0;

u_1=0.0;u_2=0.0;
y_1=0;y_2=0;

x=[0,0,0]';
error_1=0;

for k=1:1:1000
time(k)=k*ts;

rin(k)=0.5*sin(3*2*pi*k*ts);  % Tracing Sine high frequency Signal

%Linear model
yout(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;

n(k)=0.001*rands(1);          % Measure noise
yout(k)=yout(k)+n(k);         % Disturbance n(k)
```

```
error(k)=rin(k)-yout(k);

x(1)=error(k);                 % Calculating P
x(2)=(error(k)-error_1)/ts;    % Calculating D
x(3)=x(3)+error(k)*ts;         % Calculating I

kp=50;ki=0;kd=15.0;            % Tracing Sine velocity

c(k)=kp*x(1)+kd*x(2)+ki*x(3); % PID Controller

u1(k)=u_1;  %M=1, one time delay

u2(k)=(1/num1(2))*(-num1(3)*u1_1+yout(k)+den1(2)*y_1+den1(3)*y_2);

u3(k)=u2(k)-u1(k);

uo(k)=-denq(2)*uo_1-denq(3)*uo_2-denq(4)*uo_3+numq(1)*u3(k)+numq(2)*u3_1+
numq(3)*u3_2+numq(4)*u3_3;

Q=1;
if Q==0       %Not using Q(s)
    uo(k)=u3(k);
end

M=1;
if M==1       %Using Observer
   u(k)=c(k)-uo(k);
end
if M==2       %No Observer
   u(k)=c(k);
end

d(k)=50*sin(5*2*pi*k*ts);
u(k)=u(k)+d(k);  % Disturbance d(k)

if u(k)>=110          % Restricting the output of controller
   u(k)=110;
end
if u(k)<=-110
   u(k)=-110;
end
```

```
u1_1=u1(k);
u2_1=u2(k);

uo_3=uo_2;uo_2=uo_1;uo_1=uo(k);
u3_3=u3_2;u3_2=u3_1;u3_1=u3(k);

u_2=u_1;u_1=u(k);
y_2=y_1;y_1=yout(k);

error_1=error(k);
end
figure(1);
subplot(211);
plot(time,u3,'k');
xlabel('time(s)');ylabel('u3');
subplot(212);
plot(time,uo,'k');
xlabel('time(s)');ylabel('uo');
figure(2);
plot(time,d,'k',time,uo,'k');
xlabel('time(s)');ylabel('d,uo');
figure(3);
plot(time,rin,'k',time,yout,'k');
xlabel('time(s)');ylabel('rin,yout');
```

7.2 非线性系统的 PID 鲁棒控制

7.2.1 基于 NCD 优化的非线性优化 PID 控制

MATLAB 不但有用于动态系统仿真的 Simulink 工具箱，还有一个专用于非线性控制系统优化设计的工具箱 NCD（Nonlinear Control Design）。借助于 NCD 工具箱，可以实现系统参数的优化设计。

在 MATLAB NCD 工具箱中提供一个专门作系统优化设计的 NCD Blockset（非线性控制系统设计模块组），利用该模块组，系统的优化设计可以自动实现。

仿真实例

被控对象传递函数为：

$$G(s)=\frac{1.5}{50s^2+a_2s^2+a_1s+1}$$

式中，$a_2=43, a_1=3$。

系统包含饱和环节和速度限制环节±0.8 两个非线性环节。系统含有不确定因素：a_2 在 40～50 之间变化，a_1 在 $(0.5\sim2.0)\times3$ 之间变化。采用 PID 控制器，PID 的优化指标为：

（1）最大超调量不大于20%；

（2）上升时间不大于10s；

（3）调整时间不大于30s；

（4）系统具有鲁棒性。

仿真程序包括两个部分：Simulink 程序及初始化的 M 函数程序。采用 MATLAB 中的非线性系统设计工具箱 NCD 可实现 PID 控制器的优化。

仿真的关键是 NCD 功能的使用。在 Simulink 环境中双击 NCD Output 模块，弹出 NCD Blockset 约束窗口，通过 Options 菜单和 Parameters 菜单实现该功能的使用。具体说明如下：

1. Options 菜单的使用

a. 通过 Step Response 命令定义阶跃响应性能指标：

Setting time:	调整时间，选 30s
Rise time:	上升时间，选 10s
Persent setting:	稳态误差百分数，取 5
Persent overshot:	超调量百分数，取 20
Persent undershot:	振荡负幅值百分数，取 1
Step time:	启动时间，取 0
Final time:	终止时间，取 100
Initial output:	初始值，取 0
Final output:	最终值，取 1

b. 通过 Time range 命令，设置优化时间，取 0～100s

c. 通过选择 Y-Axis 命令，设置阶跃响应范围，取[0.1.5]

2. Optimization 菜单的使用

a. 选择 Parameters 项，定义调整变量及有关参数

输入：待调整优化变量 k_p,k_i,k_d 及其上下限，可取为：

下限：$\frac{k_p}{5}$　　$\frac{k_i}{5}$　　$\frac{k_d}{5}$

上限：$5k_p$　　$5k_i$　　$5k_d$

变量允差：0.001　　约束允差：0.001

b. 选择 Uncertainty 项，定义不确定变量及有关参数

输入：不确定变量 a_1,a_2 的上下限，可取为：

下限：1.5　　40

上限：6.0　　50

c. 选择 Start 命令，进行调整变量的优化，直到阶跃响应指标达到要求为止。优化时 NCD 约束窗口不断显示阶跃响应的优化过程，MATLAB COMMAND 窗口也不断显示有关信息。

每次优化结果保存在 pid_ncd.mat 文件中，以供下一次调用。阶跃响应性能限制可以直接由鼠标在 NCD Blockset 约束窗口设置。

仿真时首先运行初始化程序 chap7_3f.m，然后运行 Simulink 主程序 chap7_3.mdl（如图 7-14 所示）。优化参数如下：$k_p=1.9193,k_i=0.0978,k_d=8.1678$，响应曲线如图 7-15 所示。

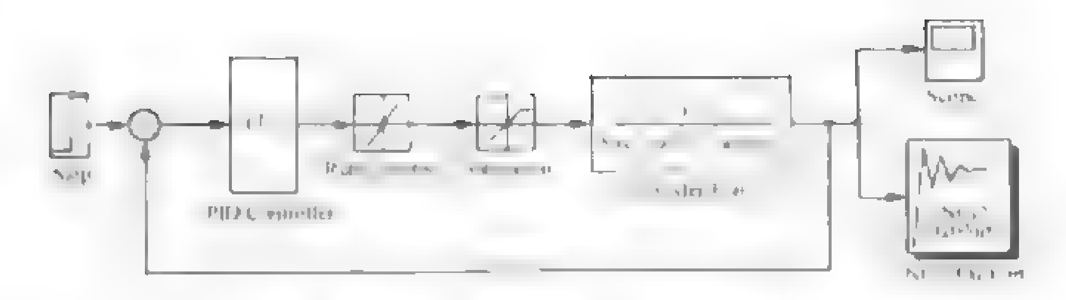


图 7-14 非线性系统 Simulink 主程序

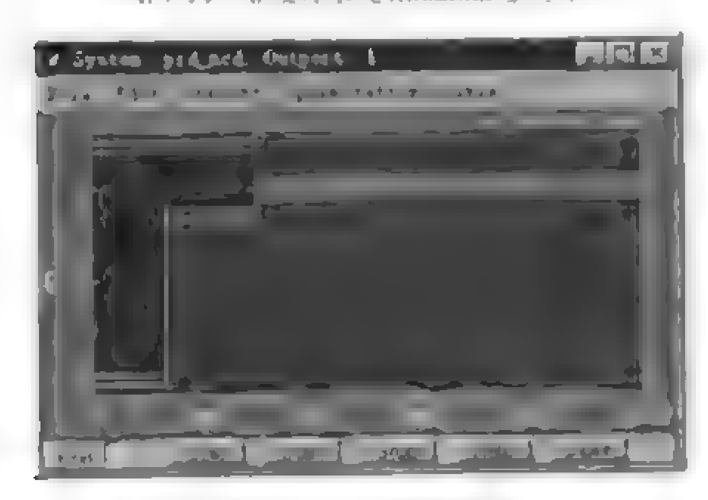


图 7-15 NCD 阶跃响应曲线图

Simulink 初始化程序：chap7_3f.m

```
[illegible]
```

Simulink 主程序：chap7_3.mdl，如图 7-14 所示

7.2.2 基于 NCD 与优化函数结合的非线性优化 PID 整定

采用 MATLAB 非线性控制系统设计工具箱的 NCD 和优化工具箱提供的各类函数可实现 PID 的整定。

在参考文献[19]中，利用 MATLAB 非线性最小二乘函数 lsqnonlin()，按照最小平方指标 $J=\int e^2 dt$ 进行 PID 参数寻优，从而实现 PID 的整定。

仿真实例

被控对象为：

$$G(s)=\frac{50(s+50)}{[illegible]}$$

运行主程序 chap7_4.m，优化结果为 $k_p=4.4124$，$k_i=0.8809$，$k_d=1.9377$。采用优化参数运行 Simulink 程序 chap7_4f2.mdl，如图 7-16 所示，优化后的阶跃响应如图 7-17 所示。

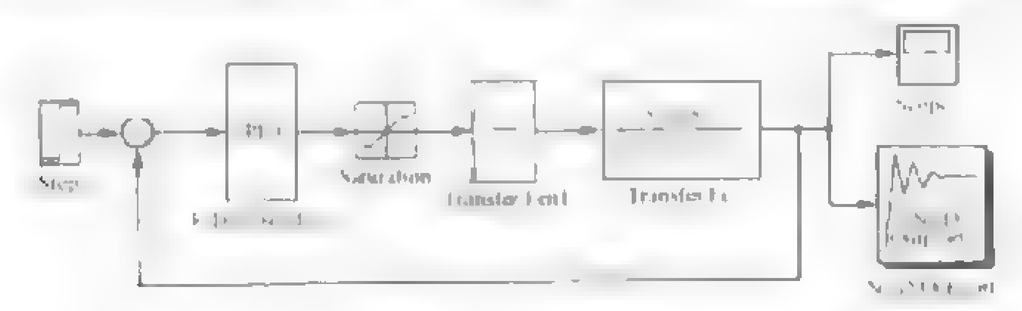


图 7-16 Simulink 程序

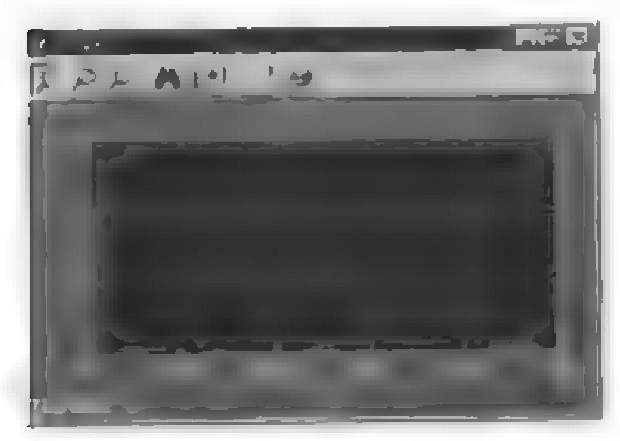


图 7-17 优化后的阶跃响应

仿真程序分为主程序、M 函数子程序和 Simulink 子程序三部分。

主程序：chap7_4.m

M 函数子程序：chap7_4f1.m

Simulink 子程序：chap7_4f2.mdl，如图 7-16 所示

7.3 一类非线性 PID 控制器设计

7.3.1 非线性控制器设计原理

设图 7-18 是一般的系统阶跃响应曲线，采用该曲线可以分析非线性 PID 控制器增益参数的构造思想。

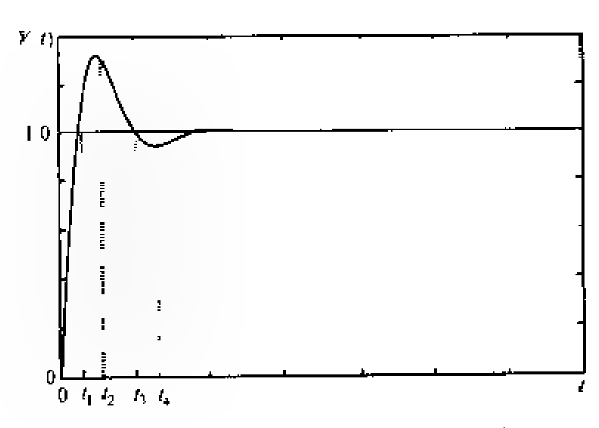


图 7-18 一般的系统阶跃响应曲线

在参考文献[22]中，非线性控制器设计原理如下：

（1）比例增益参数 k_p：在响应时间 $0<t<t_1$ 段，为保证系统有较快的响应速度，比例增益参数 k_p 在初始时应较大，同时为了减小超调量，希望误差 e_p 逐渐减小时，比例增益也随之减小；在 $t_1<t\leqslant t_2$ 段，为了增大反向控制作用，减小超调，期望 k_p 逐渐增大；在 $t_2\leqslant t\leqslant t_3$ 段，为了使系统尽快回到稳定点，并不再产生大的惯性，期望 k_p 逐渐减小；在 $t_3<t\leqslant t_4$ 段，期望 k_p 逐渐增大，作用与 $t_1\leqslant t\leqslant t_2$ 段相同。显然，按上述变化规律，k_p 随误差 e_p 变化的大致形状如图 7-19(a)所示，根据该图可以构造如下非线性函数：

$$k_p(e_p(t))=a_p+b_p(1-\operatorname{sech}(c_pe_p(t))) \tag{7.13}$$

式中，a_p,b_p,c_p 为正实常数。当误差 $e_p\to\pm\infty$ 时，k_p 取最大值为 a_p+b_p；当 $e_p=0$ 时，k_p 取最小值为 a_p；b_p 为 k_p 的变化区间，调整 c_p 的大小可调整 k_p 变化的速率。

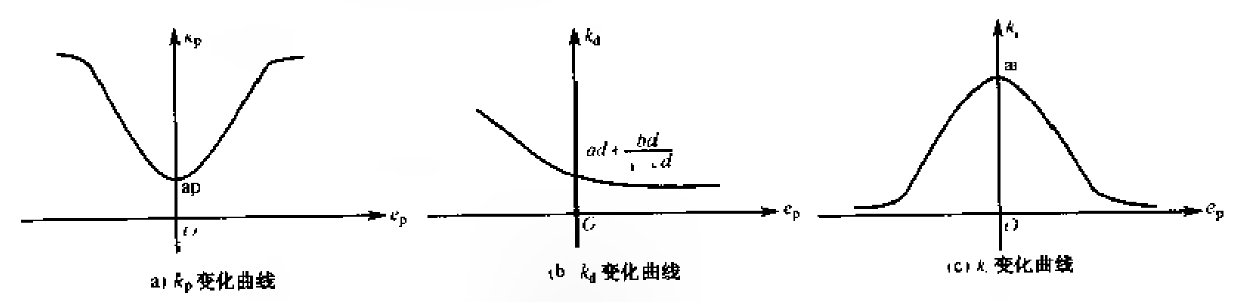


图 7-19 非线性增益调节参数变化曲线

2）微分增益参数 k_d：在响应时间 $0<t<t_1$ 段，微分增益参数 k_d 应由小逐渐增大，这样可以保证在不影响响应速度的前提下，抑制超调的产生；在 $t_1\leqslant t<t_2$ 段，继续增大 k_d，从而增大反向控制作用，减小超调量。在 t_2 时刻，减小微分增益参数 k_d，并在随后的 $t_2<t<t_4$ 段再次逐渐增大 k_d，抑制超调的产生。根据 k_d 的变化要求，在构造 k_d 的非线性函数时应考虑到误差变化速率 e_v 的符号。k_d 的变化形状如图 7-19(b)所示，所构造的非线性函数为：

$$k_d(e_p(t))=a_d+b_d/(1+c_d\exp(d_d\cdot e_p(t))) \tag{7.14}$$

式中，$e_v=\dot{e}_p$ 为误差变化速率，a_d,b_d,c_d,d_d 为正实常数，a_d 为 k_d 的最小值，a_d+b_d 为 k_d 的最大值，当 $e_p=0$ 时，$k_d=a_d+b_d/(1+c_d)$，调整 d_d 的大小可调整 k_d 的变化速率。

（3）积分增益参数 k_i：当误差信号较大时，希望积分增益不要太大，以防止响应产生振荡，有利于减小超调量；而当误差较小时，希望积分增益增大，以消除系统的稳态误差。根据积分增益的希望变化特性，积分增益参数 k_i 的变化形状如图 7-19(c) 所示，其非线性函数可表示为：

$$k_i(e_p(t))=a_i\,\mathrm{sech}(c_i e_i(t)) \tag{7.15}$$

式中，$k_i(e_p(t))=a_i\,\mathrm{sech}(c_i e_i(t))$ 为正实常数，k_i 的取值范围为 $(0,a_i)$，当 $e_p=0$ 时，k_i 取最大值。c_i 的取值决定了 k_i 的变化快慢程度。

非线性 PID 调节器的控制输入为：

$$u(t)=k_p(e_p(t))e_p(t)+k_i(e_p(t))\int_0^t e_p(t)\mathrm{d}t+k_d(e_p(t),e_v(t))\frac{\mathrm{d}e_p(t)}{\mathrm{d}t} \tag{7.16}$$

由上述分析可知，如果非线性函数中的各项参数选择适当的话。能够使控制系统既达到响应快，又无超调现象。另外，由于非线性 PID 调节器中的增益参数能够随控制误差而变化，因而其抗干扰能力也较常规线性 PID 控制强。k_p,k_i,k_d 变化的示意图如图 7-19 所示。

7.3.2 仿真程序及分析

仿真实例

求二阶传递函数的阶跃响应：

$$G_p(s)=\frac{133}{s^2+25s}$$

采用离散 PID 进行仿真，采样时间为 1ms。

取 $S=1$，针对阶跃进行仿真，阶跃响应如图 7-20 所示，其中 k_p,k_i,k_d 随偏差 $(M=1)$ 的变化曲线如图 7-21 所示；k_p,k_i,k_d 随时间 $(M=2)$ 的变化曲线如图 7-22 所示；取 $S=2$，针对正弦信号进行仿真，正弦响应如图 7-23 所示；其中 k_p,k_i,k_d 随偏差 $(M=1)$ 的变化曲线如图 7-24 所示，k_p,k_i,k_d 随时间 $(M=2)$ 的变化曲线如图 7-25 所示。

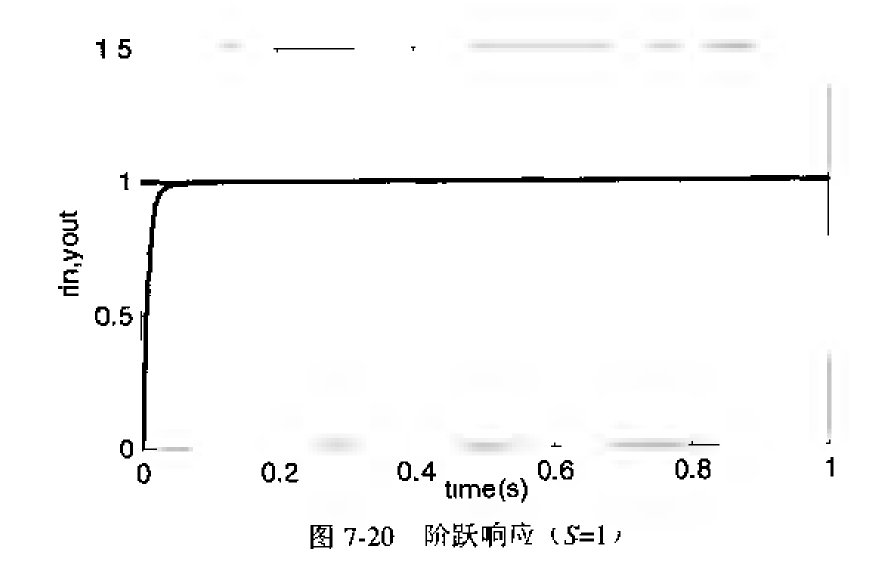


图 7-20　阶跃响应（S=1）

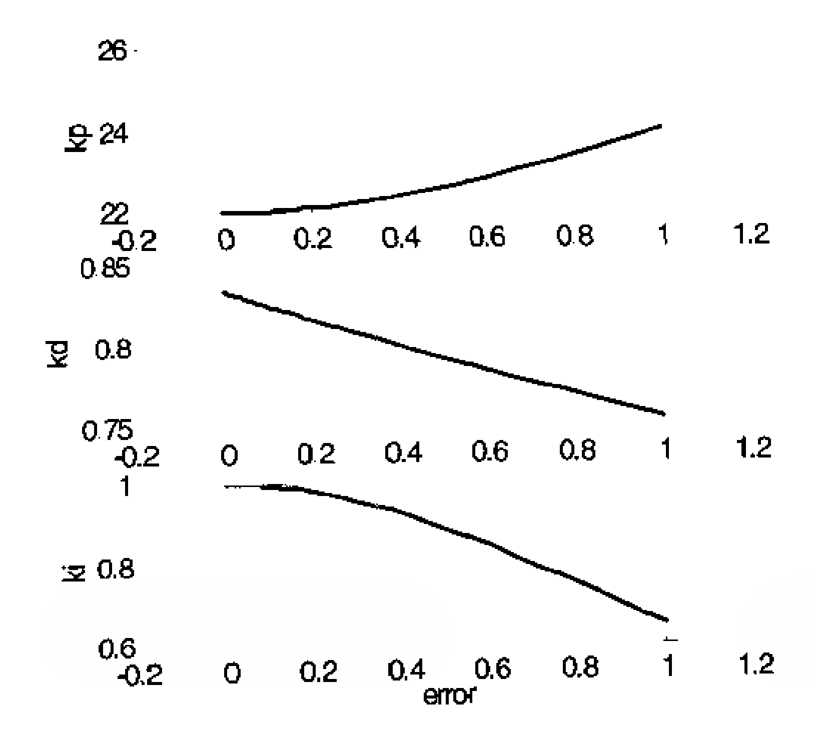


图 7-21 k_p,k_i,k_d随偏差的变化曲线（$M=1$）

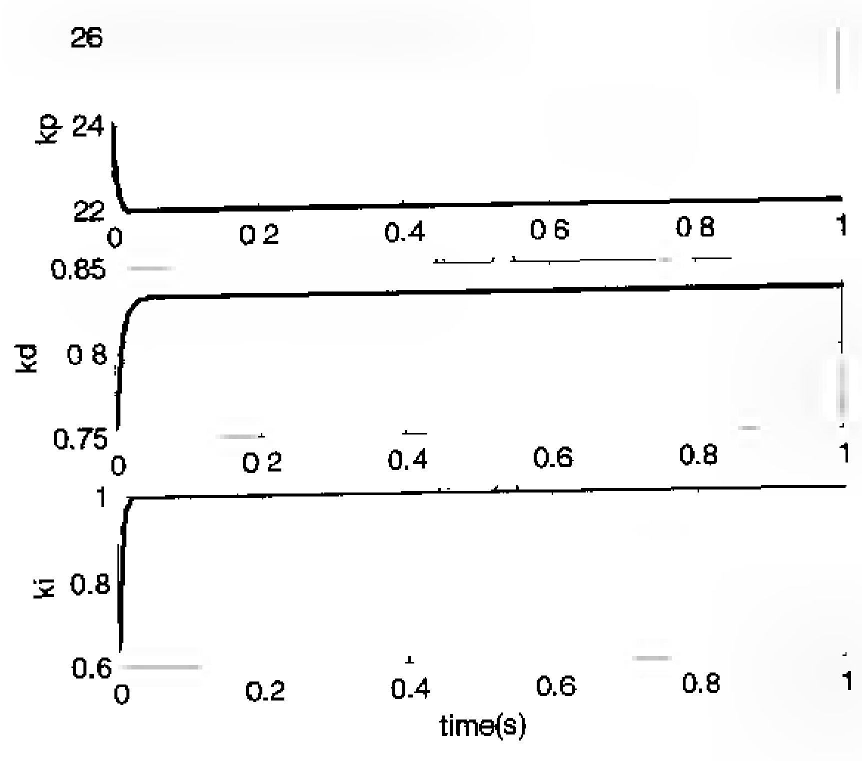


图 7-22 k_p,k_i,k_d随时间的变化曲线（$M=2$）

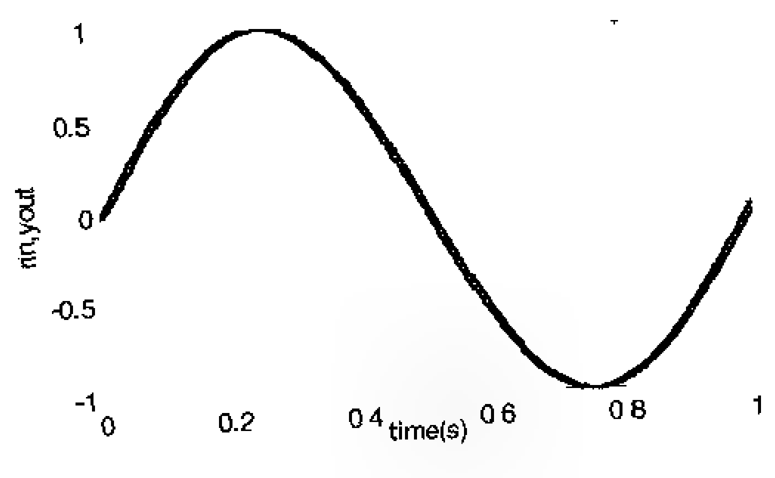


图 7-23 正弦响应（$S=2$）

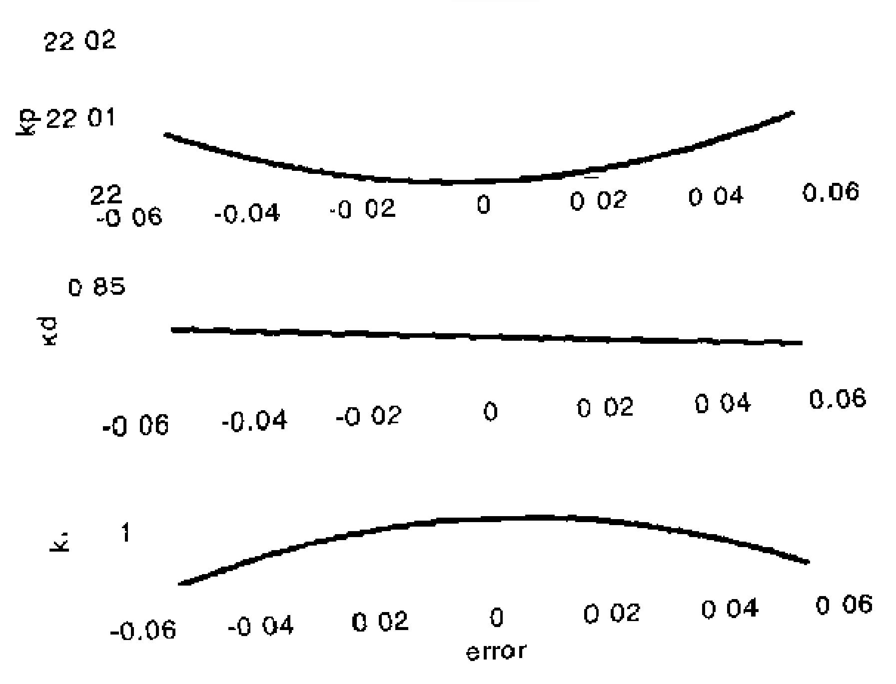


图 7-24 k_p,k_i,k_d随偏差的变化曲线（$M=1$）

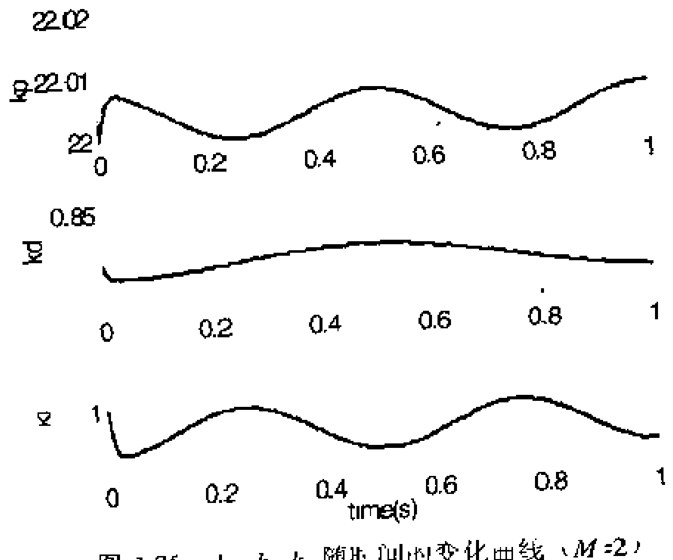


图 7-25 k_p,k_i,k_d随时间的变化曲线（$M=2$）

从仿真结果可以看出，k_p,k_i,k_d 的变化规律符合 PID 控制的原理，取得很好的仿真效果。

仿真程序：chap7_5.m。

```
%Nonlinear PID Control for a servo system
clear all;
close all;

ts=0.001;
J=1/133;
q=25/133;
sys=tf(1,[J,q,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

r_1=0;r_2=0;
u_1=0;u_2=0;
y_1=0;y_2=0;
error_1=0;
ei=0;
for k=1:1:1000
time(k)=k*ts;

S=1;
if S==1      %Step Signal
   rin(k)=1.0;
elseif S==2  %Sine Signal
   rin(k)=1.0*sin(1*2*pi*k*ts);
end

yout(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;
error(k)=rin(k)-yout(k);
derror(k)=(error(k)-error_1)/ts;

ap=22;bp=8.0;cp=0.8;
kp(k)=ap+bp*(1-sech(cp*error(k)));

ad=0.5;bd=2.5;cd=6.5;dd=0.30;
kd(k)=ad+bd/(1+cd*exp(dd*error(k)));

ai=1;ci=1;
ki(k)=ai*sech(ci*error(k));
```

```
e1=e1+error(k)*ts;
u(k)=kp(k)*error(k)+kd(k)*derror(k)+ki(k)*e1;

%Update Parameters
r_2=r_1;r_1=rin(k);
u_2=u_1;u_1=u(k);
y_2=y_1;y_1=yout(k);
error_1=error(k);
end
figure(1);
plot(time,rin,'k',time,yout,'k');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,rin-yout,'k');ylabel('error');
xlabel('time(s)');ylabel('error');
figure(3);
plot(time,derror,'k');
xlabel('time(s)');ylabel('derror');

M=1;
if M==1
   figure(4);
   subplot(311);
   plot(error,kp,'k');xlabel('error');ylabel('kp');
   subplot(312);
   plot(error,kd,'k');xlabel('error');ylabel('kd');
   ad+bd/(1+cd)
   subplot(313);
   plot(error,ki,'k');xlabel('error');ylabel('ki');
elseif M==2
   figure(5);
   subplot(311);
   plot(time,kp,'k');xlabel('time(s)');ylabel('kp');
   subplot(312);
   plot(time,kd,'k');xlabel('time(s)');ylabel('kd');
   subplot(313);
   plot(time,ki,'k');xlabel('time(s)');ylabel('ki');
end
```

7.4 基于重复控制补偿的高精度 PID 控制

7.4.1 重复控制原理

重复控制是日本的 Inoue 于 1981 年首先提出来的，用于伺服系统重复轨迹的高精度控制[23,24]。重复控制之所以能提高系统跟踪精度，其原理来源于内模原理。

加到被控对象的输入信号除偏差信号外，还叠加了一个“过去的控制偏差”，该偏差是上一周期该时刻的控制偏差。把上一次运行时的偏差反映到现在，和“现在的偏差”一起加到被控对象进行控制，这种控制方式，偏差重复被使用，称为重复控制。经过几个周期的重复控制之后可以大大提高系统的跟踪精度，改善系统品质。这种控制方法不仅适用于跟踪周期性输入信号，也可以抑制周期性干扰。

在重复控制中，一般期望重复控制作用在高频段的增益减小，为此，在重复控制中经常加入低通滤波器 $Q(s)$。本控制方法取：

$$Q(s)=\frac{1}{1+T_{q}s} \tag{7.17}$$

式中，$T_q>0$ 为滤波器的时间常数。

重复控制信号是周期性信号，其基本构造如图 7-26 所示，r 为周期参考信号。应用重复控制不仅可以提高系统的跟踪精度，还可以提高系统的鲁棒性。

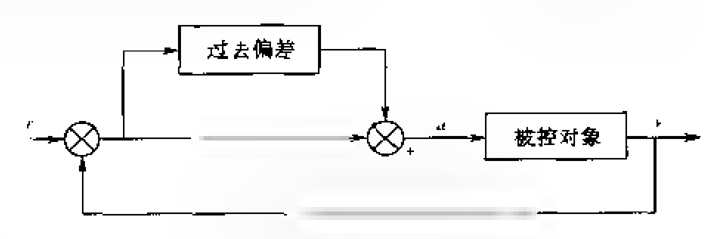


图 7-26　重复控制系统原理图

7.4.2 基于重复控制补偿的 PID 控制

基于重复控制补偿的 PID 控制系统框图如图 7-27 所示，其中 rin 为周期参考信号，周期为 L，up(k) 为 PID 控制的输出，ue(k) 为重复控制的输出。

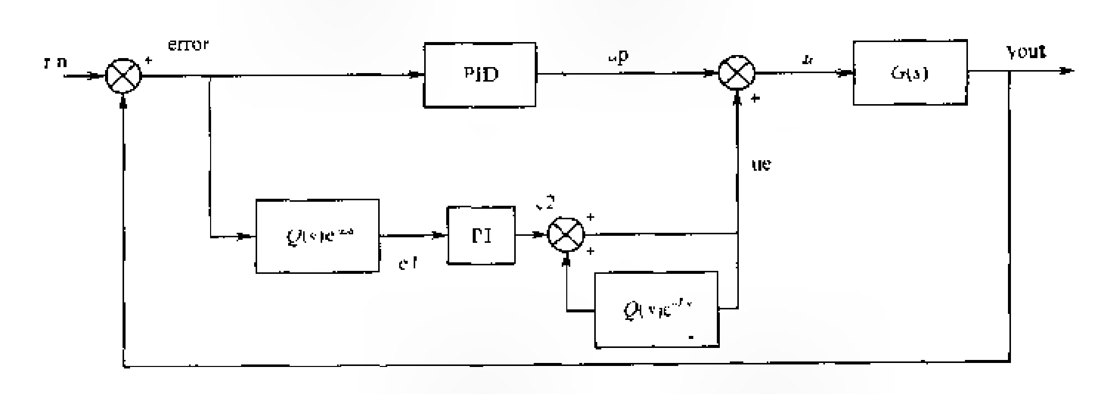


图 7-27　基于重复控制补偿的 PID 控制系统框图

控制算法为：

$$u(k)=\mathrm{up}(k)+\mathrm{ue}(k) \tag{7.18}$$

式中，$Q(s)$ 为低通滤波器。

由 $Q(s)e^{-Ls}$ 构成的正反馈回路为 $\mathrm{Gr}(z)$，$e_1(k)$为 error(k)经过 $Q(z)$后的输出，ue(k) 为 $e_2(k)$ 经过 $\mathrm{Gr}(z)$ 的输出。

7.4.3 仿真程序及分析

仿真实例一：采用 M 语言进行仿真

设被控对象为：

$$G(s)=\frac{50}{s(0.000046s^2+0.006s+1)}$$

采样时间为 ts=1ms，输入指令信号为一个正弦信号 $\mathrm{rin}(k)=1.0\sin(2\pi t)$，其中频率 $F=1.0$，幅值 $A=1.0$，则信号的重复周期 $L=\frac{1}{F\cdot \mathrm{ts}}=1000\,\mathrm{ms}$。

低通滤波器选取：

$$Q(s)=\frac{1}{0.2s+1}$$

PID 控制算法采用 PI 形式，其中 $k_\mathrm{p}=1.5, k_\mathrm{i}=10$。重复控制回路中 PI 控制算法取 $k_\mathrm{p}=2, k_\mathrm{i}=1.0$。

取 $M=1$，采用 PID 控制，位置跟踪及跟踪误差如图 7-28 和图 7-29 所示，取 $M=2$，采用重复+PID 控制，位置跟踪、跟踪误差及控制器输出如图 7-30～图 7-32 所示。

由仿真可见，采用重复控制补偿的 PID 控制可显著提高跟踪精度。

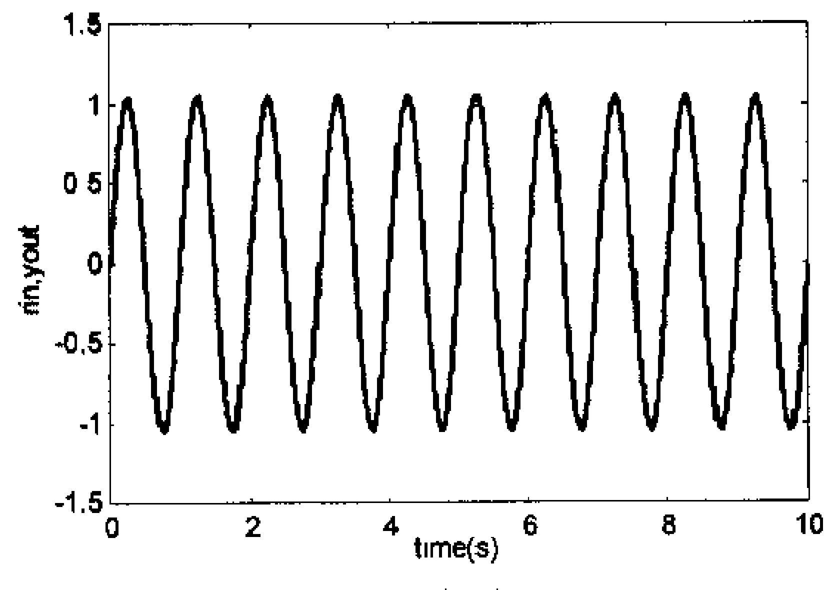


图 7-28　PID 位置跟踪　M=1

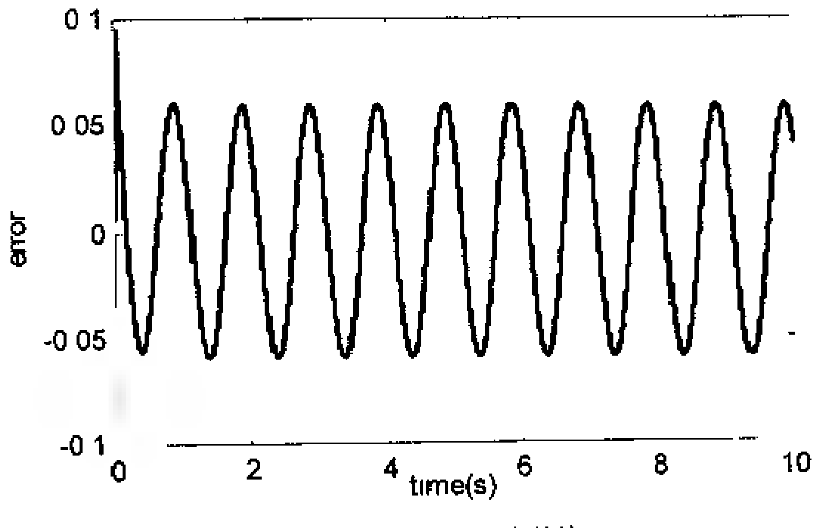


图 7-29　PID 位置跟踪误差

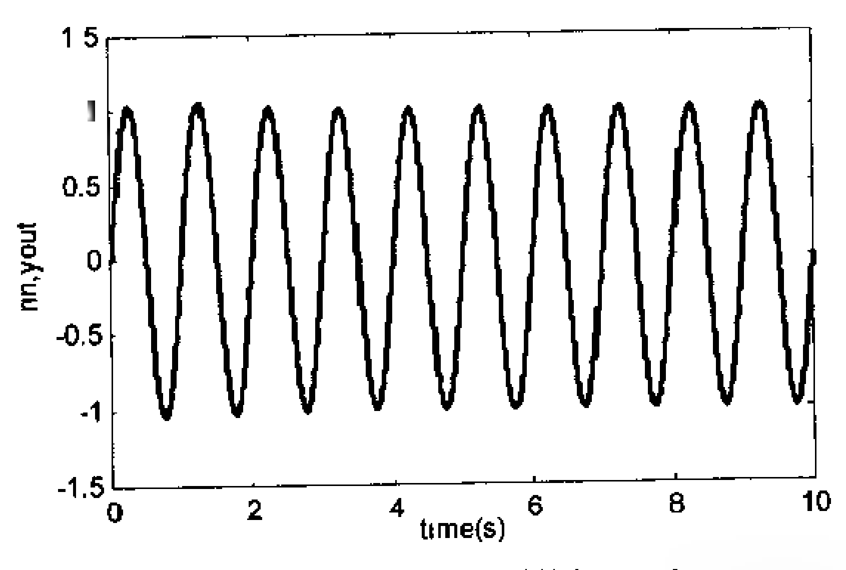


图 7-30　重复控制加 PID 位置跟踪（$M=2$）

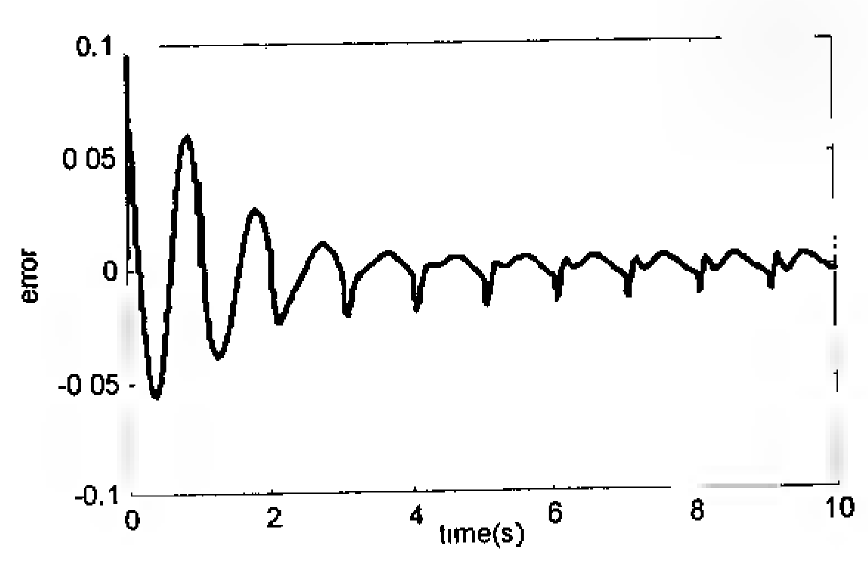


图 7-31　位置跟踪误差

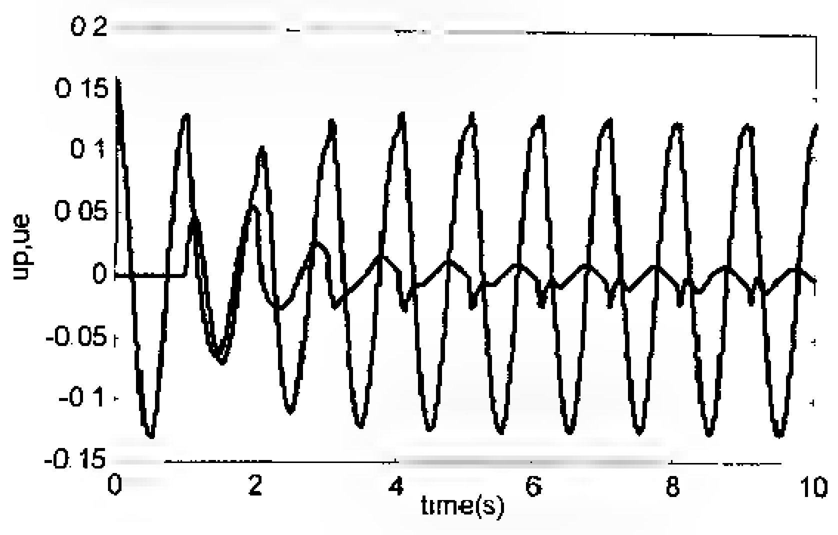


图 7-32　重复控制补偿时的控制器输出 up(*k*) 和 ue(*k*)

仿真程序 chap7_6.m。

```
%PID Control with Repetitive Control compensation
clear all;close all;

ts=0.001;
sys=tf(50,[0.000046,0.006,1,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

Q=tf(1,[0.20,1]); %Filter
dQ=c2d(Q,ts,'z');
[numQ,denQ]=tfdata(dQ,'v');

F=1;
N=1/F*1/ts;

zz=tf([1],[1 zeros(1,N)],ts);
dz=dQ*zz;
[numz,denz]=tfdata(dz,'v');

Gr=1/(1-dz);

u_1=0;u_2=0;u_3=0;
y_1=0;y_2=0;y_3=0;

ei=0;ei1=0;
ue_1=0;ue_2=0;
ue_N=0;ue_N1=0;ue_N2=0;
e2_N=0;e2_N1=0;e2_N2=0;

e_N1=0;
```

```
e1_1=0;
e2_1=0;

for k=1:1:10000
time(k)=k*ts;

rin(k)=1.0*sin(F*2*pi*k*ts);

yout(k)=-den(2)*y_1-den(3)*y_2-den(4)*y_3+num(2)*u_1+num(3)*u_2+num(4)*u_3;

e(k)=rin(k)-yout(k);

ei=ei+e(k)*ts;
up(k)=1.5*e(k)+10*ei;

e1(k)=-denQ(2)*e1_1+numQ(2)*e_N1;
ei1=ei1+e1(k)*ts;

e2(k)=2*e1(k)+1.0*ei1;

ue(k)=0.8187*ue_1+0.1813*ue_N1+e2(k)-0.8187*e2_1;

M=2;
if M==1
   u(k)=up(k);        %Only using PID
end
if M==2
   u(k)=ue(k)+up(k);  %Using REP+PID
end

if k>N
   ue_N=ue(k-N);
   e2_N=e2(k-N);
end
if k>N+1
   ue_N1=ue(k-N-1);
   e2_N1=e2(k-N-1);
   e_N1=e(k-N-1);
end
if k>N+2
   ue_N2=ue(k-N-2);
   e2_N2=e2(k-N-2);
end
```

仿真实例二：采用 Simulink 进行仿真

设被控对象为：

$$G(s)=\frac{50}{s(0.000046s^2+0.006s+1)}$$

信号滤波器选取 $Q_1(s)=\frac{1}{0.2s+1}$ 和 $Q_2(s)=\frac{1}{0.05s+1}$。

PID 控制律中采用 PI 控制，其中 $k_p=1.5, k_i=30$；重复控制律中 PI 系数取 $k_p=2, k_i=50$。采用 PID 控制时，位置跟踪误差如图 7-33 所示，采用重复+PID 控制，位置跟踪误差如图 7-34 所示。

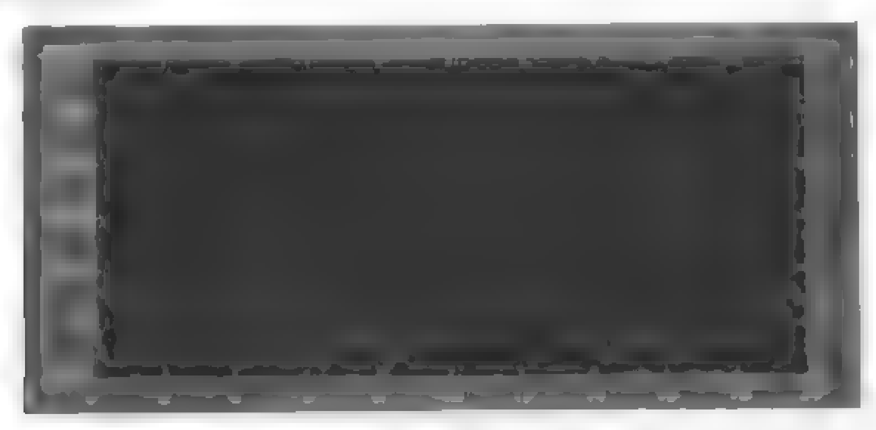


图 7-33 PID 控制跟踪误差

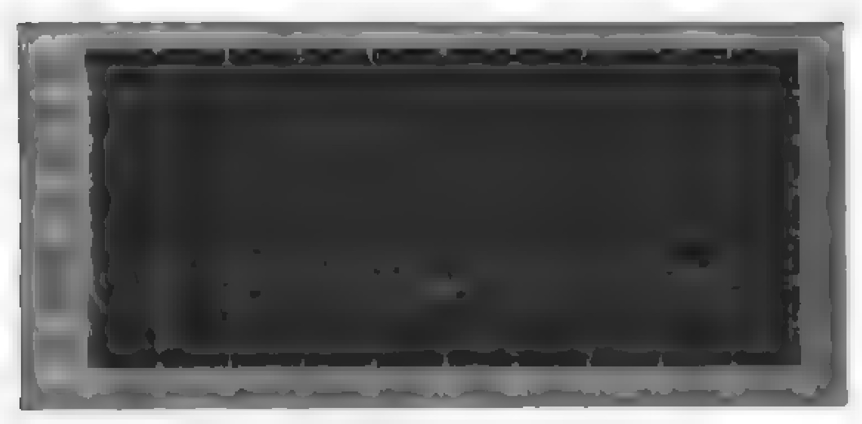


图 7-34　重复控制补偿的位置跟踪误差

仿真程序：chap7_7.mdl，如图 7-35 所示。

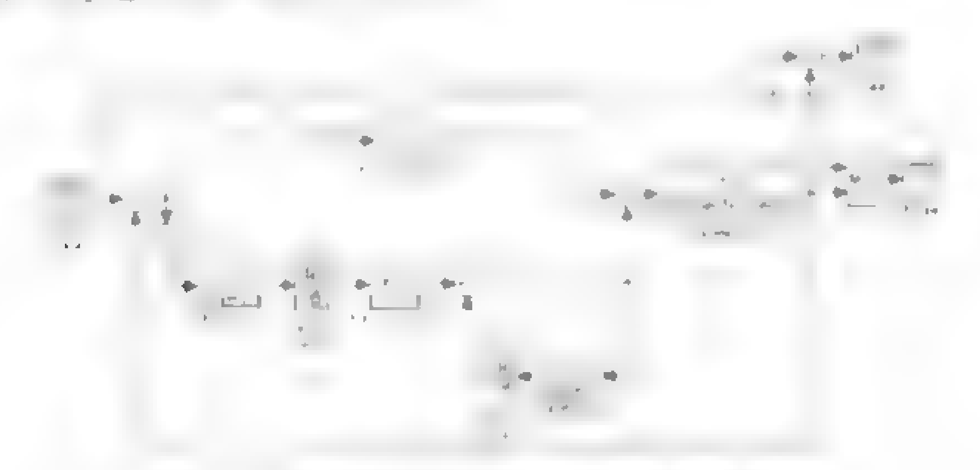


图 7-35　基于重复控制补偿的 PID 控制 Simulink 程序

7.5　基于零相差前馈补偿的 PID 控制

7.5.1　零相差控制原理

在伺服系统设计中，前馈控制可用于提高系统的跟踪性能，拓宽系统的频带。通常，前馈控制是基于不变性原理，即将前馈控制环节设计成待校正的闭环系统的逆，使校正后系统的传递函数为 1。但当闭环系统为非最小相位系统时，这种方法就不实用了。

所谓最小相位，要求所有零点都在单位圆之内。按照系统的稳定性要求闭环系统的所有极点必须在单位圆之内。由不变性原理，非最小相位系统的不稳定零点变为前馈控制器的极点，则前馈环节是不稳定的。在实际应用中，随着采样频率的提高，很多最小相位连续系统经零阶保持器离散化后所得的离散系统为非最小相位系统。零相差（Zero Phase Error）跟踪控制是一种针对非最小相位系统的数字式前馈控制器[26,27]。它通过在前馈控制器中引入零点来补偿闭环系统的不稳定零点，在指令超前值为已知时，校正后的系统在全频域范围内相移为零。

考虑如图 7-36 所示的单输入、单输出闭环 PID 控制系统，设 $G_c(z^{-1})$ 为稳定的离散对象，可视为经过的闭环系统经 ZOH 离散化而得，$G_c(z^{-1})$ 具有 d 个非最小相位零点，其静态增益为 1，则 $G_c(z^{-1})$ 可表示为：

$$G_c(z^{-1}) = \frac{z^{-d} N_u(z^{-1}) N_a(z^{-1})}{D(z^{-1})} \tag{7.19}$$

式中，$N_u(z^{-1}) = n_0 + n_1 z^{-1} + \wedge + n_s z^{-s}, n_0 \neq 0$，$N_u(z^{-1})$包含了$G_c(z^{-1})$中所有位于单位圆上及单位圆外的不稳定零点，$N_a(z^{-1})$包含了$G_c(z^{-1})$中所有位于单位圆内的稳定零点。

定理 1 设$H(z^{-1}) = N_u(z)N_u(z^{-1})$，则有：

（1）$\angle H(e^{-j\omega T}) = 0 \quad \forall \omega \in R$

（2）$\left|\angle H(e^{-j\omega T})\right|^2 = \mathrm{Re}^2\left[N_u(e^{-j\omega T})\right] + \mathrm{Im}^2\left[N_u(e^{-j\omega T})\right] \quad \forall \omega \in R$

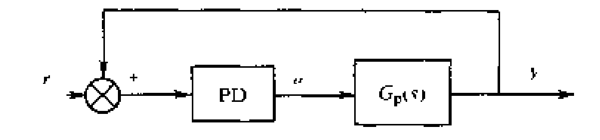


图 7-36 单输入、单输出闭环 PID 控制系统 $G_c(s)$

设零相差前馈控制器为：

$$F_u(z^{-1}) = \frac{z^{-s} N_u(z) D(z^{-1})}{N_a(z^{-1})[N_u(1)]^2} \tag{7.20}$$

则系统的实际输出为：

$$y(k) = F_u(z^{-1}) G_c(z^{-1}) r(k+s+d) = \frac{z^{-(s+d)} N_u(z) N_u(z^{-1})}{N_u^{\ 2}(1)} r(k+s+d)$$

假设

$$G(z^{-1}) = \frac{N_u(z) N_u(z^{-1})}{N_u^{\ 2}(1)} \tag{7.21}$$

显然$F_u(z^{-1})$与$G(z^{-1})$有相同的频率特性，由定理 1 可得：

$$\angle G(e^{-j\omega T}) = 0 \qquad \forall \omega \in R \tag{7.22}$$

$$G(1) = 1 \tag{7.23}$$

由此可知：如果理想输出超前值$r(k+1), \cdots, r(k+d+s)$已知，则系统在整个频域范围内相移为零，此时系统的输出为序列$\{r(k-s-d), \wedge, r(k), \wedge, r(k+s+d)\}$的移动平均值。

7.5.2 基于零相差前馈补偿的 PID 控制

基于零相差前馈的 PID 控制系统如图 7-37 所示，设计步骤如下：

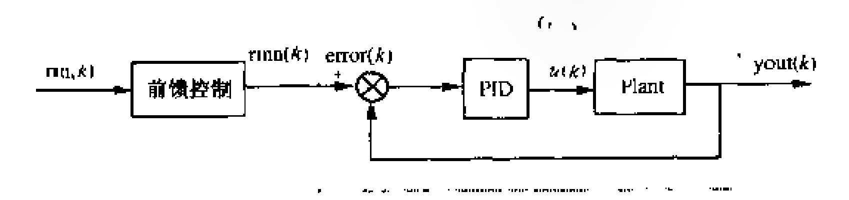


图 7-37 基于零相差前馈的 PID 控制系统结构

（1）扫频测试，得到闭环系统的幅频、相频特性。

取输入信号为正弦信号进行闭环 PID 扫频测试，测量不同频率下的相差和幅差。

设理想输入信号为：

$$rin(k)=A\sin(2\pi Ft) \tag{7.24}$$

扫频测试时频率可取为：

$$F=F_{ini}+N\times F_{step} \tag{7.25}$$

式中，F_{ini} 为扫频初始频率，F_{step} 为扫频步长。

在闭环 PID 控制器中采用 P 控制方法，可满足闭环正弦位置跟踪相差和幅差为线性，其控制律为：

$$\begin{aligned}e(k)&=rin(k)-yout(k)\\u(k)&=k_pe(k)=k_p(rin(k)-yout(k))\end{aligned} \tag{7.26}$$

通过扫频测试，并采用最小二乘法，可得到闭环系统在各个频率下的相差和幅差。最小二乘法的原理如下。

设参考信号为 $rin(k)=A_m\sin(\omega t)$，当系统为线性时，其输出角位置可表示为：

$$\begin{aligned}yout(k)&=A_f\sin(\omega t+\varphi)=A_f\cos\varphi\sin(\omega t)+A_f\sin\varphi\cos(\omega t)\\&=[\sin(\omega t)\quad\cos(\omega t)]\begin{bmatrix}A_f\cos\varphi\\A_f\sin\varphi\end{bmatrix}\end{aligned} \tag{7.27}$$

在时间域内取 $t=0,h,2h,\cdots,nh$，其中 $h=0.001,n=1000$。

设 $\boldsymbol{Y}^{\mathrm{T}}=[y(0)\quad y(h)\quad\cdots y(nh)]$，$c_1=A_f\cos\varphi$，$c_2=A_f\sin\varphi$，且有：

$$\boldsymbol{\psi}^{\mathrm{T}}=\begin{bmatrix}\sin(\omega 0) & \sin(\omega h) & \cdots\sin(\omega nh)\\\cos(\omega 0) & \cos(\omega h)\cdots & \cos(\omega nh)\end{bmatrix} \tag{7.28}$$

则可以求出 c_1,c_2 的最小二乘解为：

$$\begin{bmatrix}c_1\\c_2\end{bmatrix}=(\boldsymbol{\psi}^{\mathrm{T}}\boldsymbol{\psi})^{-1}\boldsymbol{\psi}^{\mathrm{T}}Y \tag{7.29}$$

从而得到幅频和相频特性：

$$20\lg\left(\frac{A_f}{A_m}\right)=20\lg\left(\frac{\sqrt{c_1^2+c_2^2}}{A_m}\right),\quad \varphi=\mathrm{tg}^{-1}\left(\frac{c_2}{c_1}\right) \tag{7.30}$$

（2）实现闭环系统的拟合，设计零相差控制器。

利用 Bode 图进行拟合，可得到闭环控制系统的传递函数 $G_c(s)$，通过零相差原理式（7.19）～式（7.23），可得到零相差前馈控制器。

（3）采用零相差前馈控制：

$$\begin{aligned}error(k)&=rinn(k)-yout(k)\\u(k)&=k_p error(k)\end{aligned} \tag{7.31}$$

式中，$rinn(k)$ 为前馈控制器的输出。

7.5.3 仿真程序及分析

被控对象为二阶传递函数：

$$G_p(s)=\frac{523500}{s^3+87.35s^2+10470s}$$

系统采样时间为1ms。

分为三个阶段进行仿真，分别由扫频测试(chap7_8_1.m)，闭环系统拟合及前馈控制器的设计(chap7_8_2.m)，位置跟踪(chap7_8_3.m)实现。

（1）频率测试。

运行程序之一，即扫频测试程序 chap7_8_1.m。

设位置跟踪信号为：

$$\text{rin}(k)=A\sin(2\pi Ft)$$

式中，$t=kT$，$A=0.50$。初始频率取 $F_{\text{init}}=0.50\text{Hz}$，扫频步长取 $F_{\text{step}}=0.50\text{Hz}$，取 N=15，即从 0.50～8.0Hz 进行扫频。

闭环系统采用 P 控制器，取 $k_p=7.0$。通过扫频测试，采用最小二乘法求得闭环系统的幅频和相频特性，见表 7-1。

表 7.1　扫频测试结果

F(Hz)	$20\lg(\|(A+\Delta A)/A\|)$dB	$\Delta\varphi$(°)
0.50	0.0133	5.1368
1.0	0.0524	10.2380
1.5	0.1152	15.2706
2.0	0.1984	20.2064
2.5	0.2979	25.0238
3.0	0.4087	29.7087
3.5	0.5259	34.2545
4.0	0.6443	38.6623
4.5	0.7588	42.9394
5.0	0.8648	47.0995
5.5	0.9579	51.1614
6.0	1.0343	55.1484
6.5	1.0905	59.0884
7.0	1.1234	63.0135
7.5	1.1306	66.9606
8.0	1.1099	70.9717

（2）求闭环系统的拟合传递函数，设计前馈控制器。

运行程序之二，即闭环系统拟合及前馈控制器设计程序 chap7_8_2.m。

根据表 7.1 的扫频数据，采用 MATLAB 函数 INVFREQS，对闭环系统传递函数进行拟合：

$$[B,A]=\text{INVFREQS}(H,W,\text{nb},\text{na}) \tag{7.32}$$

式中，nb 和 na 为所拟合的传递函数分子与分母的阶数，H 为在频率为 W 时理想的频率响应，B 和 A 为拟合传递函数分子与分母的系数。

取 na = 3, nb = 1，则可得闭环系统的拟合传递函数 $G_c(s)$ 为：

$$\text{sysx}(s)=\frac{-178s+3.664\times10^{5}}{s^{3}+87.49s^{2}+1.029\times10^{4}s+3.664\times10^{5}}$$

实际闭环系统传递函数和拟合传递函数sysx(*s*)的Bode图及拟合误差如图7-38和图7-39所示，将sysx(*s*)离散化得：

$$G_{c}(z^{-1})=\frac{2.665\times10^{-5}z^{2}+0.0002361z+0.0001411}{z^{3}-2.906z^{2}+2.823z-0.9162}$$

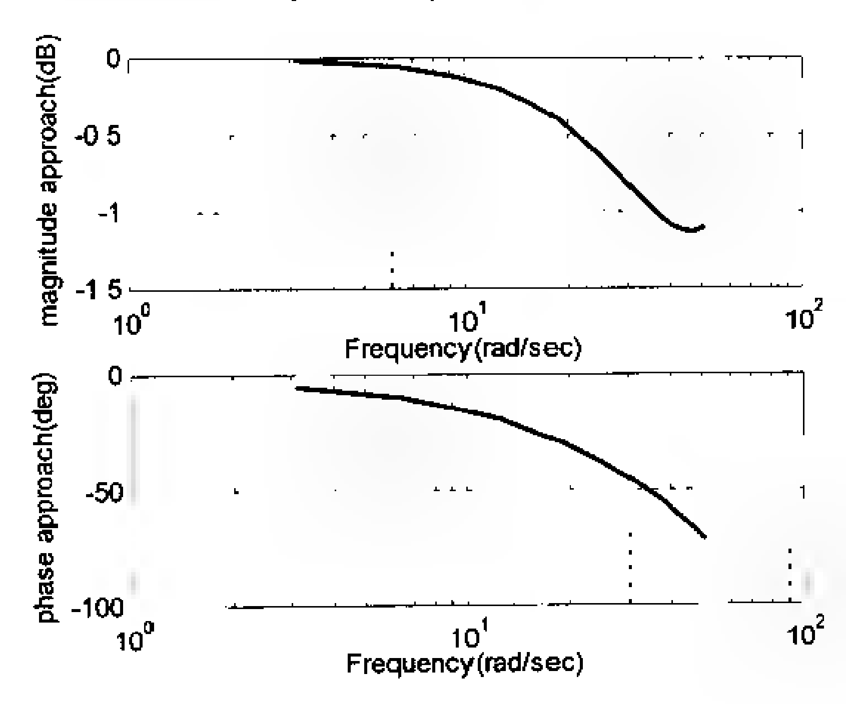


图7-38　闭环PID控制系统Bode图拟合

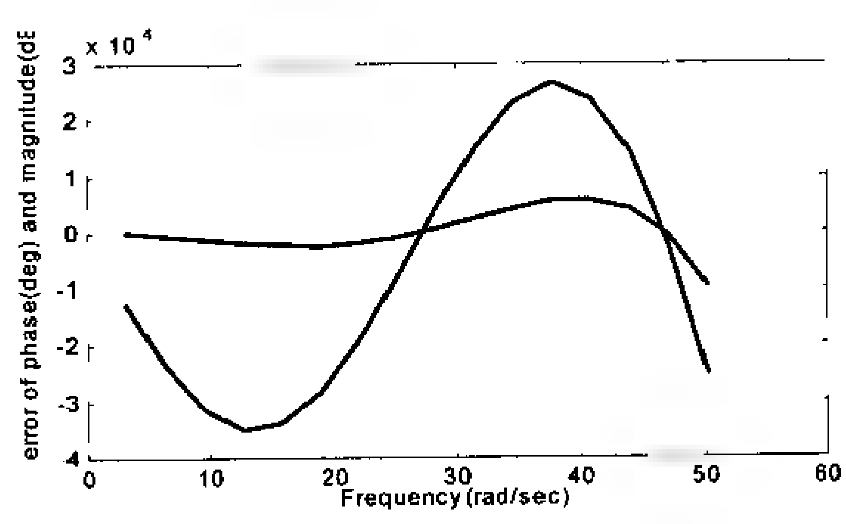


图7-39　闭环PID控制系统Bode图拟合误差

闭环系统拟合传递函数$G_{c}(z^{-1})$的零极点为：

$$z=\begin{bmatrix}-9.42221164474527\\-0.56177571156134\end{bmatrix},\quad p=\begin{bmatrix}0.97449754250883+0.086839731724651\text{i}\\0.97449754250883-0.086839731724651\text{i}\\0.95720489476102\end{bmatrix}$$

可见，$G_{c}(z^{-1})$有一个不稳定零点$z_{u}=-9.4222116447452$，将该不稳定零点转为闭环系统

拟合传递函数的极点，从而形成前馈控制器的零点，实现对闭环系统的不稳定零点的补偿，可得前馈控制器 $F_u(z^{-1})$ 为：

$$F_u(z^{-1})=\frac{4984z^4-1.501\times10^4z^3+1.561\times10^4z^2-6060z+484.7}{z+0.5618}$$

由上式可知，理想的前馈控制需要知道一个采样周期的超前值。考虑到在实际控制时理想输出超前值不知道的情况，并且采样时间很短（1ms），因此可以忽略一个采样周期的超前值，则实际的前馈控制器为：

$$F(z^{-1})=\frac{4984-1.501\times10^4z^{-1}+1.561\times10^4z^{-2}-6060z^{-3}+484.7z^{-4}}{1+0.5618z^{-1}}$$

前馈控制器输出为：

$$\begin{aligned}\mathrm{rinn}(k)=&4984\mathrm{rin}(k)-1.501\times10^4\mathrm{rin}(k-1)+1.561\times10^4\mathrm{rin}(k-2)-6060\mathrm{rin}(k-3)+\\&484.7\mathrm{rin}(k-4)-0.5618\mathrm{rinn}(k-1)\end{aligned}$$

$G_c(z^{-1})F(z^{-1})$ 的 Bode 图如图 7-40 所示，该图表明在一定频带范围内，通过前馈控制可保证在一定频带范围内实现输出高精度跟踪输入。

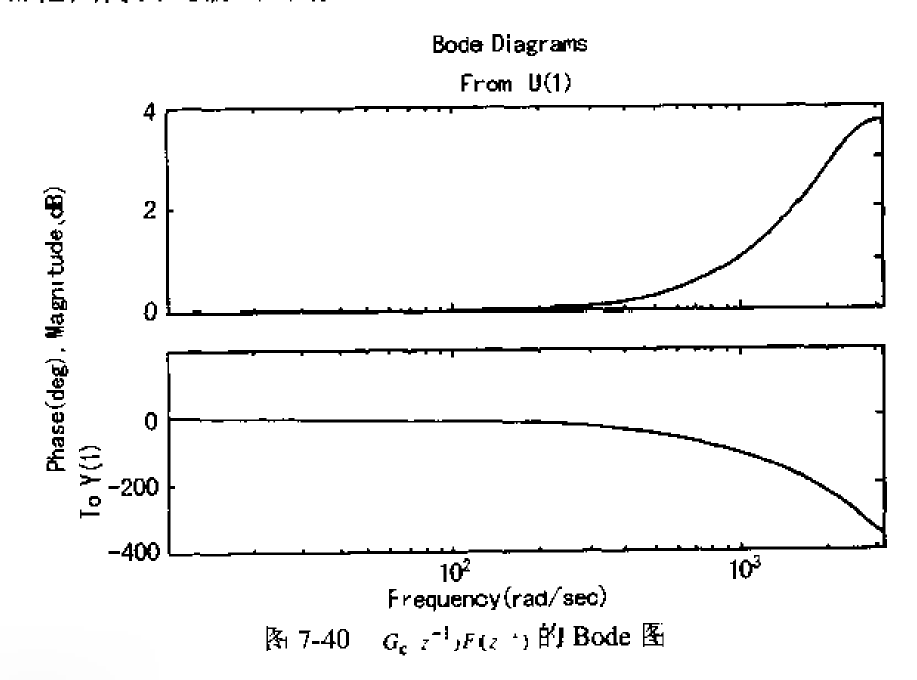


图 7-40　$G_c(z^{-1})F(z^{-1})$ 的 Bode 图

3）零相差控制器的验证。

取采样时间为 1ms，输入指令为正弦信号：

$$\mathrm{rin}(k)=0.50\sin(6\pi t)$$

假设超前信号已知，通过将 $G_c(z^{-1})F_u(z^{-1})$ 最小化并进行零极点对消可得系统的输入、输出关系为：

$$\mathrm{yout}(k)=-0.1328\mathrm{rin}(k+2)+1.266\mathrm{rin}(k+1)-0.1328\mathrm{rin}(k)$$

假设超前信号未知，对输入信号取一个采样时间延迟，通过将 $G_c(z^{-1})F(z^{-1})$ 最小化并进行零极点对消可得系统的输入、输出关系为：

$$\mathrm{yout}(k)=-0.1328\mathrm{rin}(k-1)+1.266\mathrm{rin}(k-2)-0.1328\mathrm{rin}(k-3)$$

通过上述输入、输出关系可以对零相差控制器进行验证，其中 $M=1$ 时为超前信号已知，正弦跟踪结果如图 7-41 和图 7-42 所示；$M=2$ 时为超前信号未知，正弦跟踪结果如图 7-43

和图 7-44 所示。仿真结果表明，当超前信号未知时，位置跟踪性能下降。

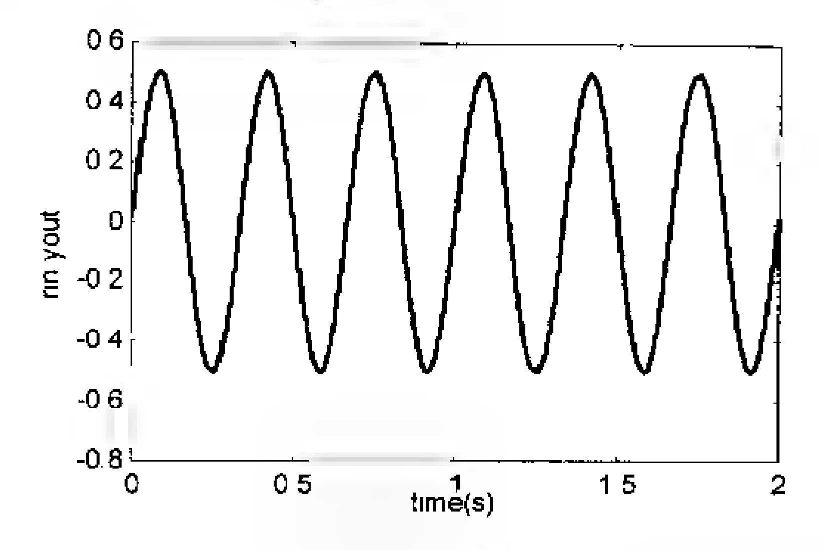


图 7-41 正弦跟踪（M=1）

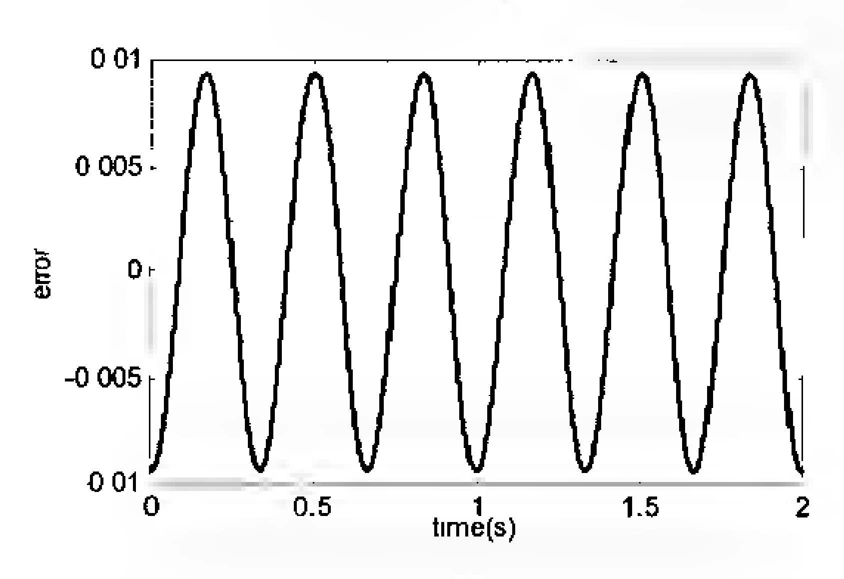


图 7-42 正弦跟踪误差（M=1）

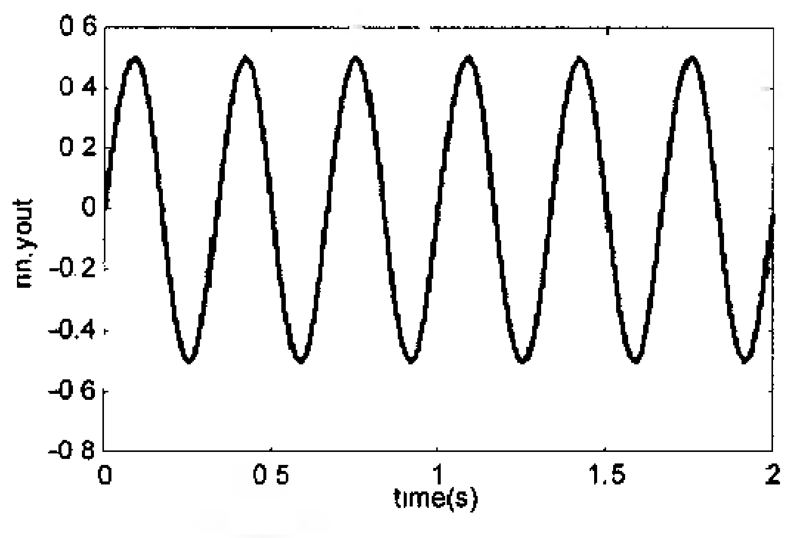


图 7-43 正弦跟踪（M=2）

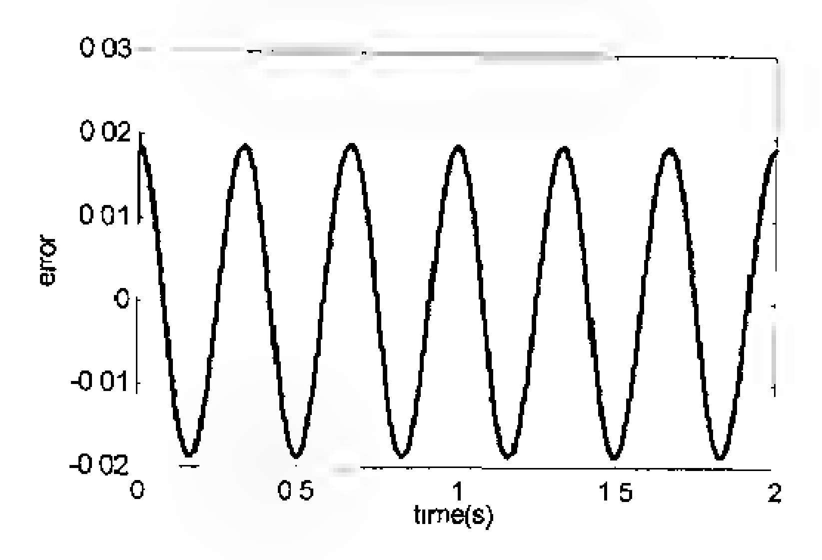


图 7-44　正弦跟踪误差（M=2）

零相差控制器验证程序：chap7_8_0.m。

```
%Zero Phase Error controller verify
clear all;
close all;
ts=0.001;

rin_5=0;rin_4=0;rin_3=0;rin_2=0;rin_1=0;
F=3;
G=2000;
for k=1:1:G
time(k)=k*ts;
rin(k)=0.50*sin(F*2*pi*k*ts);

M=2;
if M==1
    rin(k+3)=0.50*sin(F*2*pi*(k+3)*ts);
    rin(k+2)=0.50*sin(F*2*pi*(k+2)*ts);
    rin(k+1)=0.50*sin(F*2*pi*(k+1)*ts);
    yout(k)=0.1328*rin(k+2)+1.266*rin(k+1)+0.1328*rin(k);
elseif M==2
    yout(k)=0.1328*rin_1+1.266*rin_2+0.1328*rin_3;
end

rin_5=rin_4;rin_4=rin_3;rin_3=rin_2;rin_2=rin_1;rin_1=rin(k);
end
figure(1);
plot(time(1:1:G),rin(1:1:G),'r',time(1:1:G),yout(1:1:G),'b');
figure(2);
plot(time(1:1:G),rin(1:1:G)-yout(1:1:G),'r');
```

（4）实现位置跟踪。

运行程序之三，即位置跟踪程序 chap7_8_3.m。

采用基于零相差前馈补偿的PID控制进行正弦位置跟踪，闭环仍采用扫频时的P控制。当 $S=1$ 时，输入指令为 $rin(k)=0.50\sin(3\pi Ft)$，取 $F=3.0$，正弦位置跟踪如图 7-45 所示。当 $S=2$ 时，输入指令为 $rin(k)=0.50\sin(2\pi t)+1.0\sin(6\pi t)+1.0\sin(10\pi t)$，正弦叠加信号位置跟踪如图 7-46 所示。

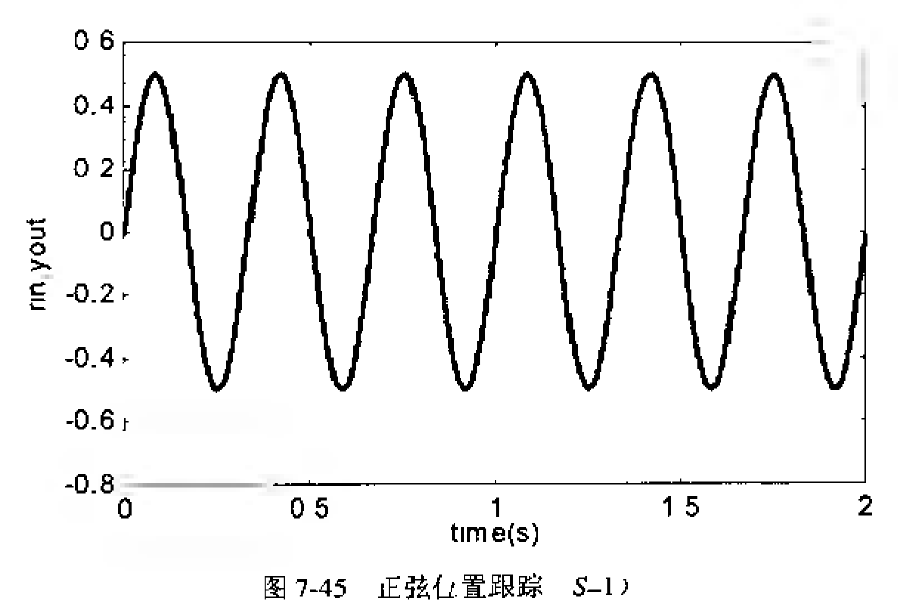


图 7-45　正弦位置跟踪（$S=1$）

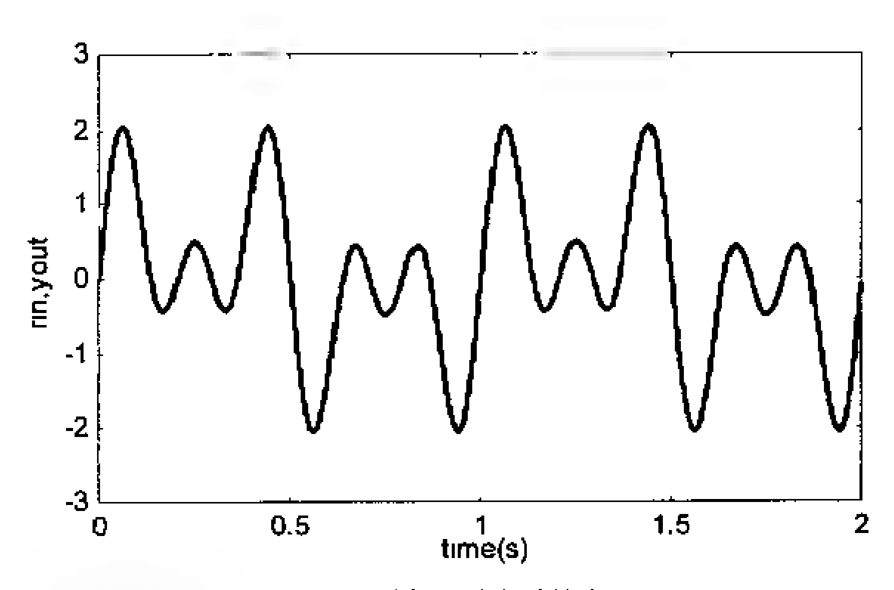


图 7-46　正弦叠加信号位置跟踪（$S=2$）

程序之一：扫频测试 chap7_8_1.m。

```
%Zero Phase Error Frequency testing
clear all;
close all;

ts=0.001;
sys=tf(5.235e005,[1,87.35,1.047e004,0]);
```

```
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

kp=0.70;
error_1=0;
kk=0;       %Frequency steps

for F=0.5:0.5:8
kk=kk+1;
FF(kk)=F;

u_1=0.0;u_2=0.0;u_3=0.0;
y_1=0;y_2=0;y_3=0;
for k=1:1:2000
time(k)=k*ts;

%Tracing Sine high frequency Signal
rin(k)=0.5*sin(1*2*pi*F*k*ts);

%Linear model
yout(k)=-den(2)*y_1-den(3)*y_2-den(4)*y_3+num(2)*u_1+num(3)*u_2+num(4)*u_3;

error(k)=rin(k)-yout(k);
u(k)=kp*error(k);    %P Controller

%---------Return of PID parameters---------%
u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);
end

plot(time,rin,'r',time,yout,'b');
pause(0.0000000000001);

Y=rin(1001:1:2000)';

for i=1:1:1000
    fai(1,i)=sin(2*pi*F*i*ts);
    fai(2,i)=cos(2*pi*F*i*ts);
end

c=inv(fai*fai')*fai*Y;
```

```
pinA=sqrt(c(1)*c(1)+c(2)*c(2));
pinF=atan(c(1)/c(2));

Yplot=y(1:2000);

for i=1:10
    fai(1,i)=sin(2*pi*F*i*ts);
    fai(2,i)=cos(2*pi*F*i*ts);
end

c=inv(fai*fai')*fai*Yplot';

poutA=sqrt(c(1)*c(1)+c(2)*c(2));
poutF=atan(c(2)/c(1));

mag(kk)=20*log10(poutA/pinA);    %Magnitude
ph(kk)=(poutF-pinF)*180/pi;      %Phase error

end
FF=FF';
mag=mag';
ph=ph';

save freq.mat FF mag ph;
save closed1.mat kp;
```

程序之二：闭环系统拟合及前馈控制器设计 chap7_8_2.m。

```
%Close system approaching and zero phase error controller design
clear all;
close all;
load freq.mat;

ts=0.001;
f=FF;
for i=1:1:length(ph)
   if ph(i)>0 ph(i)=ph(i)-360;
   end
end

%Transfer function approaching
%(1)Freq parameters
w=2*pi*f;             %From Hz to rad/sec
```

```
magl=10.^(mag/20);   %From dB to degree
phl=ph*pi/180;       %From degree to radian

h=magl.*cos(phl)+j*magl.*sin(phl);

% 2 Continuous function
na=3;   %Three tanks approaching
nb=1;
%bb and aa are real numerator and denominator of transfer function
[bb,aa]=invfreqs(h,w,nb,na);   %w contains the frequency values in radians/s
display('Transfer function approaching is:');
sysx=tf(bb,aa)
[zs,ps,ks]=zpkdata(sysx,'v');

% 3 Discrete function
Gc=c2d(sysx,ts,'zoh');
zpksys=zpk(Gc);
[z,p,k]=zpkdata(zpksys,'v');   %Getting zero pole gain: z,p,k
display('Zeros and Poles of the Transfer function is');
z
p
%In z-1 format
zGc=tf(Gc);
[nGc,dGc]=tfdata(zGc,'v');
zGc=filt(nGc,dGc,ts);

%(4 Magnitude and Phase to draw Bode
%Frequency response,create the complex frequency response vector: h=a+bi
h=freqs(bb,aa,w);
%Magnitude and phase
sysmag=abs(h);                %Degree
sysmagl=20*log10(sysmag);      %From degree to dB
sysph=angle(h);               %Get radian
sysphl=sysph*180/pi;           %From radian to degree

% 5 Drawing practical plant and its approach function Bode to compare
figure(1);
subplot(2,1,1);
semilogx(w,sysmagl,'r',w,mag,'b');grid on;
xlabel('Frequency(rad/sec)');ylabel('magnitude approach(dB)');

subplot(2,1,2);
```

```
semilogx(w,sysph1,'r',w,ph,'b');grid on;
xlabel('Frequency(rad/sec)');ylabel('phase approach(deg)');

figure(2);
magError=sysmag1-mag;
phError=sysph1-ph;
plot(w,phError,'r',w,magError,'b');
xlabel('Frequency(rad/sec)');ylabel('error of phase(deg) and magnitude(dB)');

%(6)Plant in zero-pole-gain format
zu=z(1);   %Unstable zero point
z1=z(2);
p1=p;
p1(4)=1/zu;

k1=1;
Gctemp=zpk(z1,p1,k1,ts);
dc=dcgain(Gctemp);         %Getting DC gain
k1=1/dc;                   %C(1)=1;

Gcn=zpk(z1,p1,k1,ts);

%(7)Design controller
Fdz=1/Gcn;       %Fdz=zpk(p1,z1,1/k1,ts);
display('ZPE Controller is:');
tfdz=tf(Fdz)      %z^(-3):three rank delay

[nn1,dd1]=tfdata(tfdz,'v');

nF=nn1;
dF(1)=dd1(4);    %z^(-3):three rank delay
dF(2)=dd1(5);
dF(3)=dd1(1);
dF(4)=dd1(2);
dF(5)=dd1(3);

format long;
display('LPF Controller coefficient is:');
nF
dF
save zpecoeff.mat nF dF;
```

```
%Controller
F filt nF,dF,ts ;   %Equal conversion:from z to z
dcgain F ;

% 8 Verify the controller
Gn series F,Gc ;    %Gn z  F z *Gc z
figure 3 ;
bode Gn ;
yk minreal Gn,ts ;  %Sampling Gn z
```

程序之 ：位置跟踪 chap7 8_3.m。

```
%Zero Phase Error Position control
clear all;
close all;
load zpecoeff.mat;  %ZPE coefficient nF and dF
load closed.mat;    %Load kp

ts 0 001;
sys tf 5 235e005 [1,8 35,1 047e004,0] ;
dsys c2d sys,ts  z' ;
[num,den] tfdata dsys,'v' ;

u_1 0.0;u_2 0.0;u_3 0 0;
rin_5 ;rin_4 0;rin_3 0;rin 2 0;rin_1 0;
rinn 1 0;
y_1 0,y 2 0;y 3 0;
error 1 0,

F 3;
S 2,
for k 1:1:2000
time k  k*ts;

if S 1          %Sine Signal
    rin k  0.50*sin F*2*pi*k*ts ;
elseif S  2     %Random Signal
    rin k  0.50*sin 1*2*pi*k*ts +1. *sin 3*2*pi*k*ts +1 0*sin 5*2*pi*k*ts ,
end

rinn k  nF 1 *rin k +nF 2 *rin_1+nF 3 *rin 2+nF 4 *rin_3+nF 5 *rin 4
        dF 2 *rinn ;
```

```
%Linear model
yout(k)=-den(2)*y_1-den(3)*y_2-den(4)*y_3+num(2)*u_1+num(3)*u_2+num(4)*u_3;
error(k)=rin(k)-yout(k);
u(k)=kp*error(k);    %P Control

u_3=u_2;u_2=u_1;u_1=u(k);
rin_5=rin_4;rin_4=rin_3;rin_3=rin_2;rin_2=rin_1;rin_1=rin(k);
rinn_1=rinn(k);
y_3=y_2;y_2=y_1;y_1=yout(k);
end
figure(1);
plot(time,rin,'r',time,yout,'b');
xlabel('time(s)');ylabel('rin,yout');
figure(2);
plot(time,rin-yout,'r');
xlabel('time(s)');ylabel('error');
```

7.6 基于卡尔曼滤波器的 PID 控制

7.6.1 卡尔曼滤波器原理

在现代随机最优控制和随机信号处理技术中，信号和噪声往往是多维非平稳随机过程。因其时变性，功率谱不固定。在 1960 年初提出了卡尔曼滤波理论，该理论采用时域上的递推算法在数字计算机上进行数据滤波处理。

对于离散域线性系统：

$$\begin{aligned} x(k) &= Ax(k-1) + B(u(k) + w(k)) \\ y_v(k) &= Cx(k) + v(k) \end{aligned} \tag{7.33}$$

式中，$w(k)$ 为过程噪声信号，$v(k)$ 为测量噪声信号。

离散卡尔曼滤波器递推算法为：

$$M_n(k) = \frac{P(k)C^T}{CP(k)C^T + R} \tag{7.34}$$

$$P(k) = AP(k-1)A^T + BQB^T \tag{7.35}$$

$$P(k) = (I_n - M_n(k)C)P(k) \tag{7.36}$$

$$x(k) = Ax(k-1) + M_n(k)(y_v(k) - CAx(k-1)) \tag{7.37}$$

$$y_e(k) = Cx(k) \tag{7.38}$$

误差的协方差为：

$$\mathrm{errcov}(k) = CP(k)C^T \tag{7.39}$$

卡尔曼滤波器结构如图 7-47 所示。

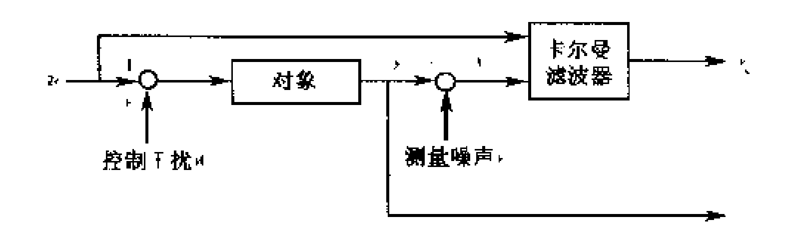


图 7-47 卡尔曼滤波器结构

7.6.2 仿真程序及分析

仿真实例

验证卡尔曼滤波器的滤波性能。

对象为二阶传递函数：

$$G_{\mathrm{p}}(s)=\frac{133}{s^2+25s}$$

取采样时间为 1ms，采用 Z 变换将对象离散化，并描述为离散状态方程的形式：

$$x(k+1)=\boldsymbol{A}x(k)+\boldsymbol{B}(u(k)+w(k))$$

$$y(k)=\boldsymbol{C}x(k)$$

带有测量噪声的被控对象输出为：

$$y_v(k)=\boldsymbol{C}x(k)+v(k)$$

式中，$\boldsymbol{A}=\begin{bmatrix}1.0000000,0.0009876\\0.0000000,0.9753099\end{bmatrix}$，$\boldsymbol{B}=\begin{bmatrix}0.0000659\\0.1313512\end{bmatrix}$，$\boldsymbol{C}=[1,0]$，$\boldsymbol{D}=[0]$

仿真方法一：采用 M 语言进行仿真

控制干扰信号 $w(k)$ 和测量噪声信号 $v(k)$ 幅值均为 0.10 的白噪声信号，输入信号幅值为 1.0、频率为 1.5Hz 的正弦信号，采用卡尔曼滤波器实现信号的滤波，取 $Q=1$，$R=1$。仿真时间为 3s，原始信号及带有噪声的原始信号、原始信号及滤波后的信号和误差协方差的变化分别如图 7-48～图 7-50 所示。仿真结果表明，该滤波器对控制干扰和测量噪声具有很好的滤波作用。

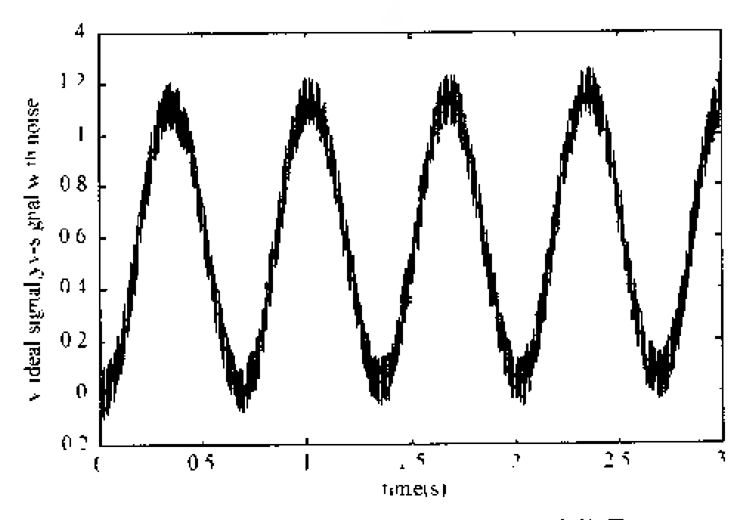


图 7-48 原始信号及带有噪声的原始信号

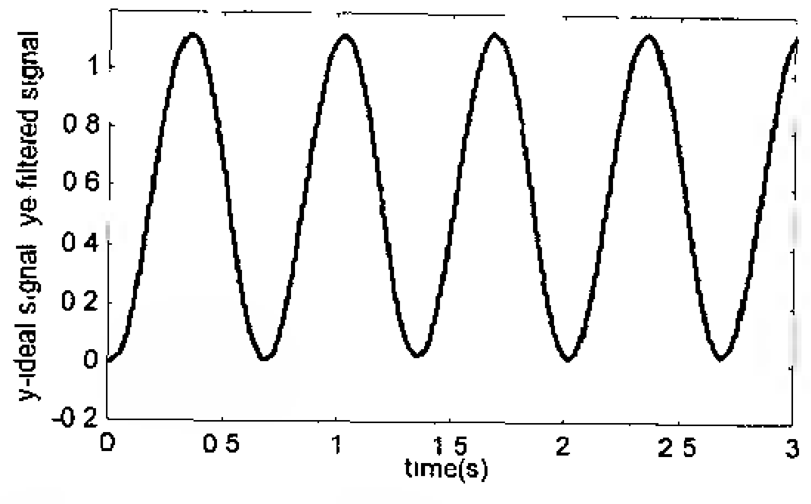


图 7-49 原始信号及滤波后的信号

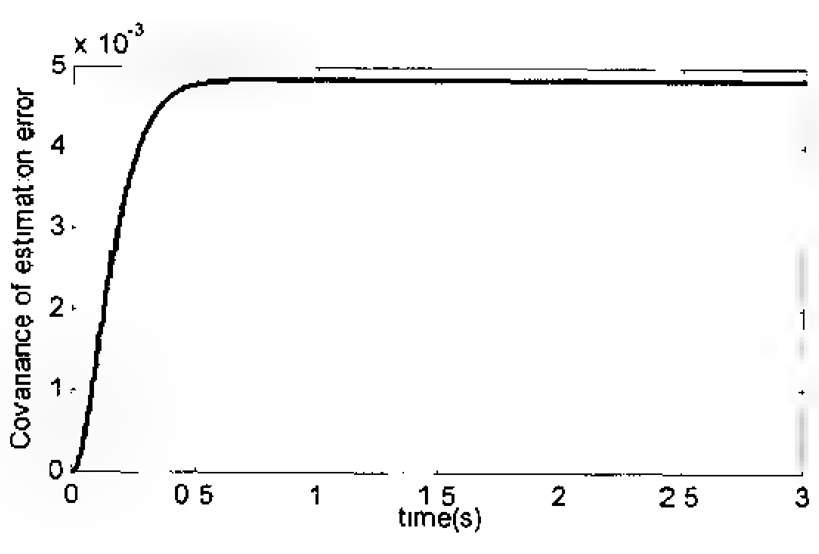


图 7-50 误差协方差的变化

仿真程序：chap7_9.m。

```
%Kalman filter
%x=Ax+B(u+w(k));
%y=Cx+D+v(k)
clear all;
close all;

ts=0.001;
M=3000;

%Continuous Plant
a=25;b=133;
sys=tf(b,[1,a,0]);
dsys=c2d(sys,ts,'z');
```

```
[num,den]=tfdata(dsys,'v');

A1=[0 1;-0 -a1];
B1=[0;b];
C1=[1 0];
D1=[0];
[A,B,C,D]=c2dm(A1,B1,C1,D1,ts,'z');

Q=1;            %Covariances of w
R=1;            %Covariances of v

P=B*Q*B';          %Initial error covariance
x=zeros(2,1);      %Initial condition on the state

ye=zeros(M,1);
ycov=zeros(M,1);

u_1=0;u_2=0;
y_1=0;y_2=0;

for k=1:1:M
time(k)=k*ts;

w(k)=0.10*rands(1);   %Process noise on u
v(k)=0.10*rands(1);   %Measurement noise on y

u(k)=1.0*sin(2*pi*1.5*k*ts);
u(k)=u(k)+w(k);

y(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;
yv(k)=y(k)+v(k);

%Measurement update
   Mn=P*C'/(C*P*C'+R);
   P=A*P*A'+B*Q*B';
   P=(eye(2)-Mn*C)*P;

   x=A*x+Mn*(yv(k)-C*A*x);
   ye(k)=C*x+D;          %Filtered value
```

仿真方法二：采用 Simulink 进行仿真

Kalman 滤波[illegible]干扰信号 $w(k)$ 和观测噪声信号 $v(k)$ 幅值均为 0.10 的白噪声信号，[illegible]为 1.0、频率为 0.5Hz 的正弦信号，[illegible]滤波器实现信号的滤波，[illegible] $Q=1$，$R=1$，[illegible] 10s，[illegible] 7-51 和图 7-52 所示。

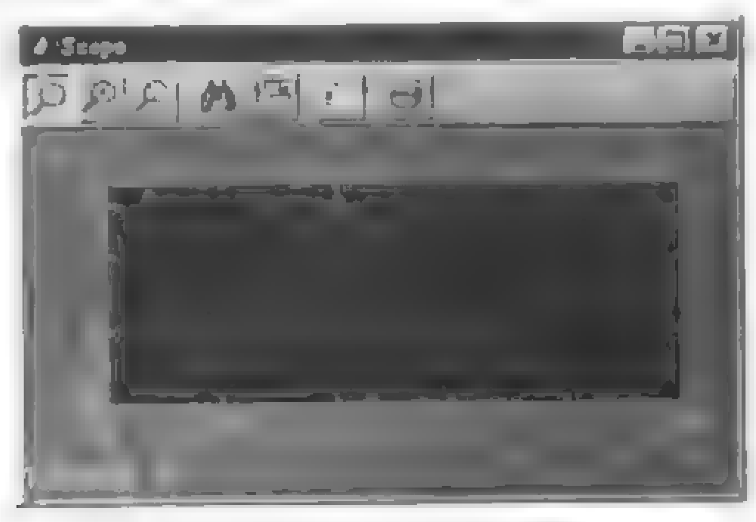


图 7-51 [illegible]信号 y 及滤波后的信号 y_e

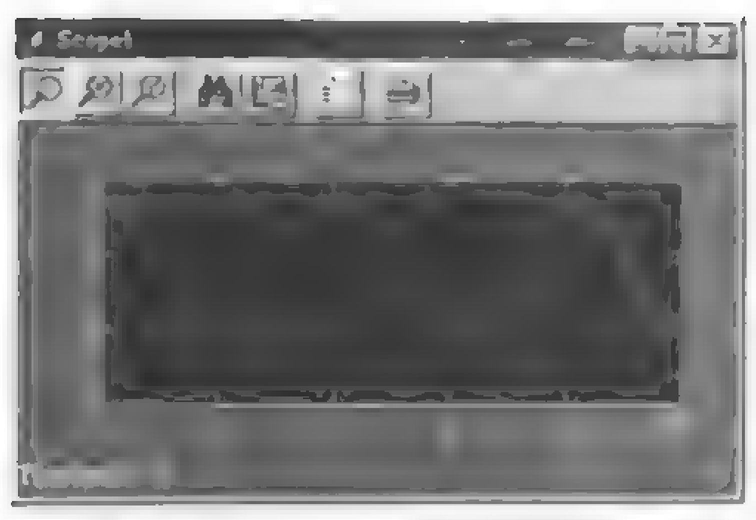


图 7-52 原始信号 y 及带有噪声的原始信号

Simulink 主程序：chap7_10.mdl，仿真程序如图 7-53 所示。

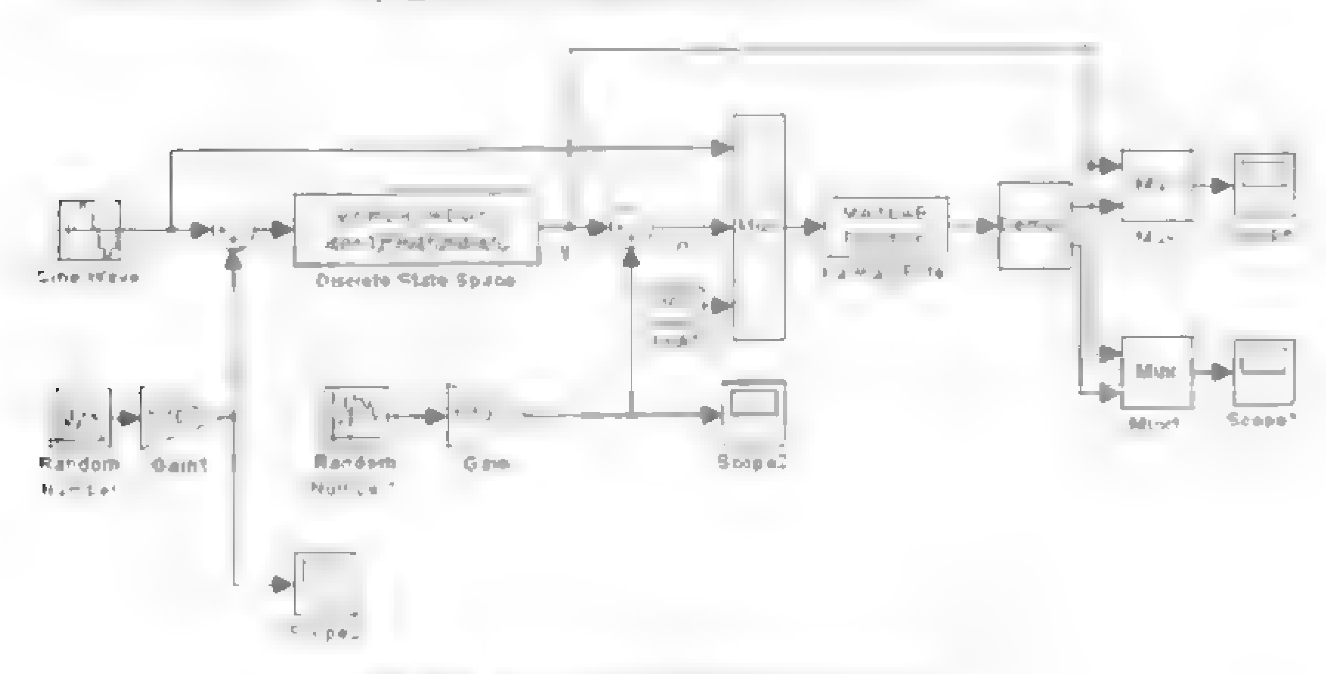


图 7-53 基于 Kalman 滤波器的 Simulink 仿真

Kalman 滤波子程序：chap7_10f.m

```
    sys=tf(b,[1,a,0]);
    A1=[0 1;0 -a];
    B1=[0;b];
    C1=[1 0];
    D1=[0];
    [A,B,C,D]=c2dm(A1,B1,C1,D1,ts,'z');

     Q=1;                %Covariances of w
     R=1;                %Covariances of v
    P=B*Q*B';           %Initial error covariance
end

%Measurement update
Mn=P*C'/(C*P*C'+R);

x=A*x+Mn*(yv-C*A*x);

P=(eye(2)-Mn*C)*P;

ye=C*x+D;            %Filtered value

u(1)=ye;
u(2)=yv;

errcov=C*P*C';       %Covariance of estimation error

%Time update
x=A*x+B*u1;
P=A*P*A'+B*Q*B';
```

7.6.3 基于卡尔曼滤波器的PID控制

基于卡尔曼（Kalman）滤波器的PID控制系统结构如图7-54所示。

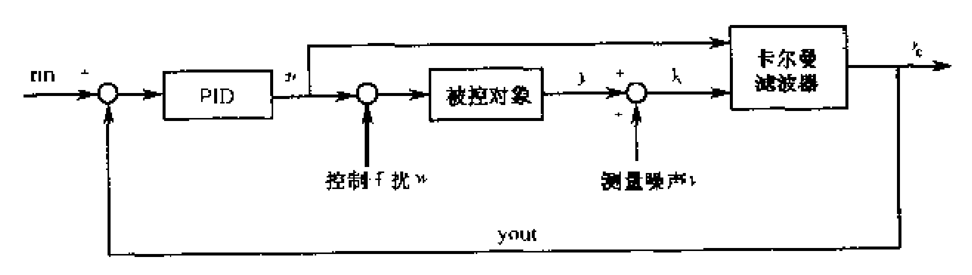


图7-54　基于卡尔曼滤波器的PID控制系统结构

7.6.4 仿真程序及分析

仿真实例

采用卡尔曼滤波器的PID控制。

被控对象为二阶传递函数：

$$G_p(s)=\frac{133}{s^2+25s}$$

离散化结果与7.6.2节的仿真实例相同，采样时间为1ms。

控制干扰信号 $w(k)$ 和测量噪声信号 $v(k)$ 幅值均为0.002的白噪声信号，输入信号为一阶跃信号，采用卡尔曼滤波器实现信号的滤波，取 $Q=1$，$R=1$，仿真时间为1s。分两种情况进行仿真：$M=1$ 时为未加滤波，$M=2$ 时为加滤波。在PID控制器中，取 $k_p=8.0, k_i=0.80, k_d=0.20$。加入滤波器前后PID阶跃响应如图7-55和图7-56所示，仿真结果表明，采用滤波器使控制效果明显改善。

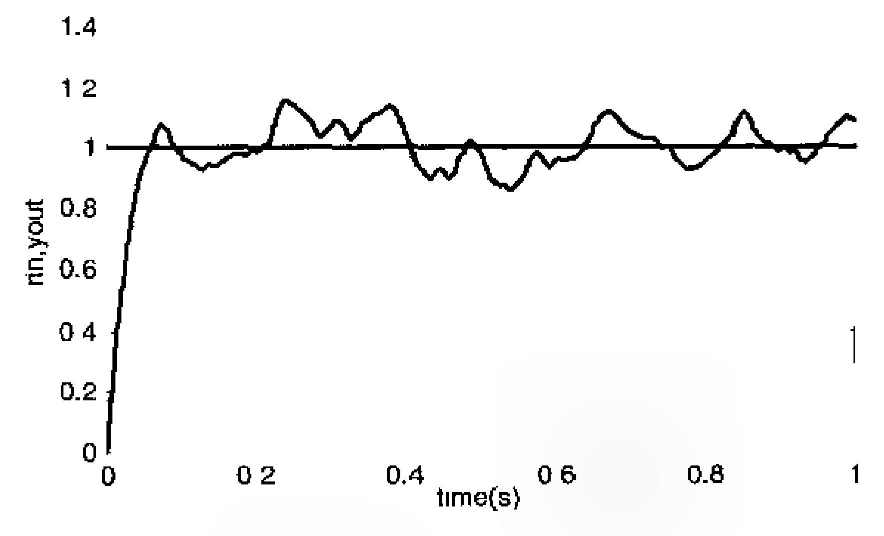


图7-55 无滤波器时PID控制阶跃响应（M=1）

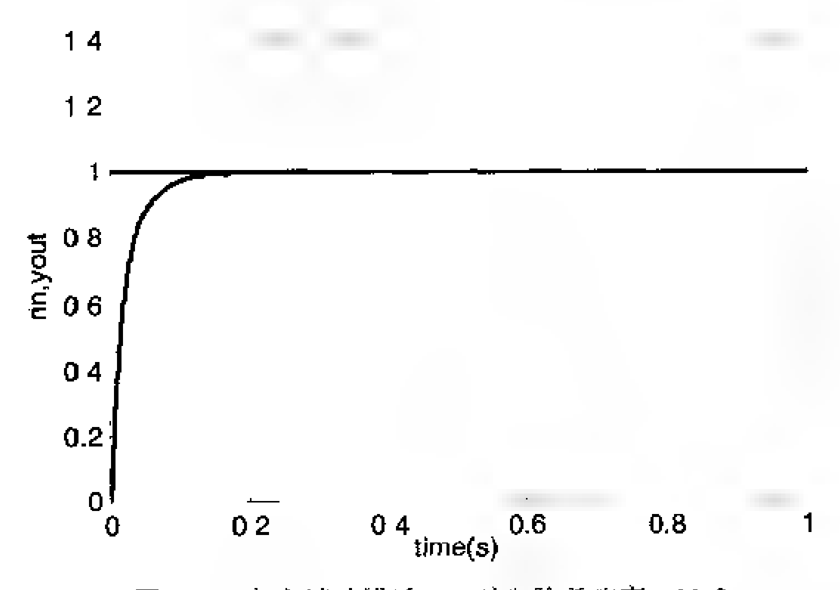


图7-56 加入滤波器后PID控制阶跃响应（M=2）

仿真程序：chap7_11.m

```
%Discrete Kalman filter for PID control
%Reference kalman_2rank.m
%x=Ax+B(u+w(k));
%y=Cx+D+v(k)
clear all;
close all;

ts=0.001;
%Continuous Plant
a=25;b=133;
sys=tf(b,[1 a,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

A1=[0 1;0 -a];
B1=[0;b];
C1=[1 0];
D1=[0];
[A,B,C,D]=c2dm(A1,B1,C1,D1,ts,'z');

Q=1;               %Covariances of w
R=1;               %Covariances of v

P=B*Q*B';           %Initial error covariance
x=zeros(2,1);       %Initial condition on the state

u_1=0;u_2=0;
y_1=0;y_2=0;
ei=0;
error_1=0;
for k=1:1:1000
time(k)=k*ts;

rin(k)=1;
kp=8.0;ki=0.80;kd=0.20;

w(k)=0.002*rands(1);   %Process noise on u
v(k)=0.002*rands(1);   %Measurement noise on y

y(k)=-den(2)*y_1-den(3)*y_2+num(2)*u_1+num(3)*u_2;
```

```
yv(k)=y(k)+v(k);

%Measurement update
Mn=P*C'/(C*P*C'+R);
P=A*P*A'+B*Q*B';
P=(eye(2)-Mn*C)*P;

x=A*x+Mn*(yv(k)-C*A*x);
ye(k)=C*x+D;      %Filtered value

M=1;
if M==1              %Not using filter
    yout(k)=yv(k);
elseif M==2          %Using filter
    yout(k)=ye(k);
end
error(k)=rin(k)-yout(k);
ei=ei+error(k)*ts;

u(k)=kp*error(k)+ki*ei+kd*(error(k)-error_1)/ts;     %PID
u(k)=u(k)+w(k);

errcov(k)=C*P*C';       %Covariance of estimation error

%Time update
x=A*x+B*u(k);

u_2=u_1;u_1=u(k);
y_2=y_1;y_1=yout(k);
error_1=error(k);
end
figure(1);
plot(time,rin,'k',time,yout,'k');
xlabel('time(s)');
ylabel('rin,yout');
```

7.7 单级倒立摆的 PID 控制

7.7.1 单级倒立摆建模

倒立摆系统的控制问题一直是控制研究中的一个典型问题。控制的目标是通过给小车底

座施加一个力u（控制量），使小车停留在预定的位置，并使杆不倒下，即不超过一个预先定义好的垂直偏离角度范围。图 7-57 为一级倒立摆系统示意图，小车质量为M，摆的质量为m，小车位置为x，摆的角度为θ。

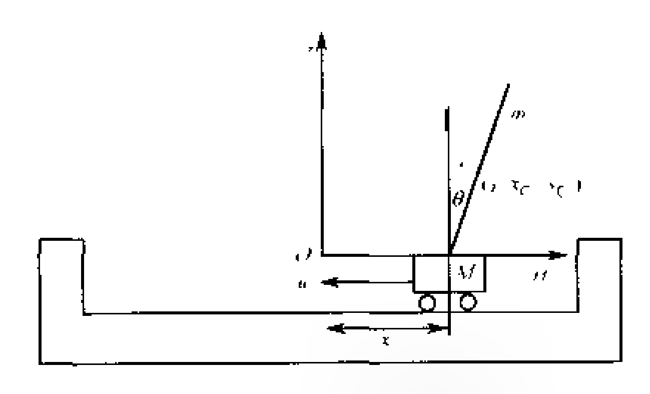


图 7-57　一级倒立摆系统示意图

设摆杆偏离垂直线的角度为θ，同时规定摆杆重心的坐标为(x_G, y_G)，则有：

$$x_G = x + l\sin\theta$$

$$y_G = l\cos\theta$$

根据牛顿定律，建立水平和垂直运动状态方程。

摆杆围绕其重心的转动运动可用力矩方程来描述：

$$I\ddot{\theta} = Vl\sin\theta - Hl\cos\theta$$

式中，I为摆杆围绕其重心的转动惯量。

摆杆重心的水平运动由下式描述：

$$m\frac{\mathrm{d}^2}{\mathrm{d}t^2}(x + l\sin\theta) = H$$

摆杆重心的垂直运动由下式描述：

$$m\frac{\mathrm{d}^2}{\mathrm{d}t^2}l\cos\theta = V - mg$$

小车的水平运动由下式描述：

$$M\frac{\mathrm{d}^2 x}{\mathrm{d}t^2} = u - H$$

假设θ很小，$\sin\theta \approx \theta$，$\cos\theta \approx 1$。则以上各式变为：

$$I\ddot{\theta} = Vl\theta - Hl \tag{7.40}$$

$$m(\ddot{x} + l\ddot{\theta}) = H \tag{7.41}$$

$$0 = V - mg \tag{7.42}$$

$$M\ddot{x} = u - H \tag{7.43}$$

由式（7.41）和式（7.43）得：

$$(M + m)\ddot{x} + ml\ddot{\theta} = u \tag{7.44}$$

由式（7.40）和式（7.42）得：

$$(I + ml^2)\ddot{\theta} + ml\ddot{x} = mgl\theta \tag{7.45}$$

由式（7.44）和式（7.45）可得单级倒立摆方程：

$$\ddot{\theta}=\frac{m(m+M)gl}{(M+m)I+Mml^2}\theta-\frac{ml}{(M+m)I+Mml^2}u \tag{7.46}$$

$$\ddot{x}=-\frac{m^2gl^2}{(M+m)I+Mml^2}\theta+\frac{I+ml^2}{(M+m)I+Mml^2}u \tag{7.47}$$

式中，$I=\frac{1}{12}mL^2$，$l=\frac{1}{2}L$

控制指标共有 4 个，即单级倒立摆的摆角θ、摆速$\dot{\theta}$、小车位置x和小车速度$\dot{x}$。将倒立摆运动方程转化为状态方程的形式。令$x(1)=\theta$，$x(2)=\dot{\theta}$，$x(3)=x$，$x(4)=\dot{x}$，则式（7.46）和式（7.47）可表示为状态方程式（7.48）。

$$\dot{x}=\boldsymbol{A}x+\boldsymbol{B}u \tag{7.48}$$

式中，$\boldsymbol{A}=\begin{pmatrix}0&1&0&0\\t_1&0&0&0\\0&0&0&1\\t_2&0&0&0\end{pmatrix}$，$\boldsymbol{B}=\begin{pmatrix}0\\t_3\\0\\t_4\end{pmatrix}$，$t_1=\frac{m(m+M)gl}{(M+m)I+Mml^2}$，$t_2=-\frac{m^2gl^2}{(M+m)I+Mml^2}$，

$t_3=-\frac{ml}{(M+m)I+Mml^2}$，$t_4=\frac{I+ml^2}{(M+m)I+Mml^2}$

7.7.2 单级倒立摆控制

对每个控制目标都选取 PD 控制方式，控制器为：

$$u_i(k)=k_{\mathrm{p}i}\mathrm{e}i(k)+k_{\mathrm{d}i}\mathrm{de}i(k) \tag{7.49}$$

式中，$\mathrm{e}i(k)$ 和 $\mathrm{de}i(k)$ 为控制指标 i 的误差和误差变化率。

$$u(k)=\sum_{i=1}^{4}u_i(k) \tag{7.50}$$

为了进行对比，采用最优控制中的 LQR 方法。该方法针对状态方程 $\dot{x}=\boldsymbol{A}x+\boldsymbol{B}u$，通过确定最佳控制量$u(t)=-\boldsymbol{K}x(t)$的矩阵$\boldsymbol{K}$，使得控制性能指标$J=\int_0^\infty(x^{\mathrm{T}}\boldsymbol{Q}x+u^{\mathrm{T}}\boldsymbol{R}u)\mathrm{d}t$达到极小，其中$\boldsymbol{Q}$为正定（或半正定）厄米特或实对称矩阵，$\boldsymbol{R}$为正定厄米特或实对称矩阵，$\boldsymbol{Q}$和$\boldsymbol{R}$分别表示了误差和能量损耗的相对重要性，$\boldsymbol{Q}$中对角矩阵的各个元素分别代表各项指标误差的相对重要性。LQR 控制器的增益为：

$$\boldsymbol{K}=\mathrm{LQR}(\boldsymbol{A},\boldsymbol{B},\boldsymbol{Q},\boldsymbol{R}) \tag{7.51}$$

$$u(k)=-\boldsymbol{K}x \tag{7.52}$$

7.7.3 仿真程序及分析

仿真中倒立摆的参数为：$g=9.8\mathrm{m/s^2}$（重力加速度），$M=1.0\mathrm{kg}$（小车质量），$m=0.1\mathrm{kg}$（杆的质量），$L=0.5\mathrm{m}$（杆的半长），$\mu_{\mathrm{c}}=0.0005$（小车相对于导轨的摩擦系数），$\mu_{\mathrm{p}}=0.000002$（杆相对于小车的摩擦系数）。$F$ 为作用于小车上的力，即控制器的输出，在$[-10,+10]$上连续取值。

采样周期$T=20\mathrm{ms}$，初始条件取$\theta(0)=-10°$，$\dot{\theta}(0)=0$，$x(0)=0.20$，$\dot{x}(0)=0$，期望状态

为：$\theta(0)=0, \dot{\theta}(0)=0$，$x(0)=0, \dot{x}(0)=0$，其中摆动角度值应转变为弧度值

取 $S=1$，采用 PID 控制。由于电机控制方向必须与摆的倒动角度方向相反，故控制器参数选负数，PID 控制参数选取见主程序。使用 PID 时倒立摆响应结果及控制器输出如图 7-58 和图 7-59 所示，可见，采用 PID 控制可实现单级倒立摆的控制，但 PID 控制器的参数较难选取。

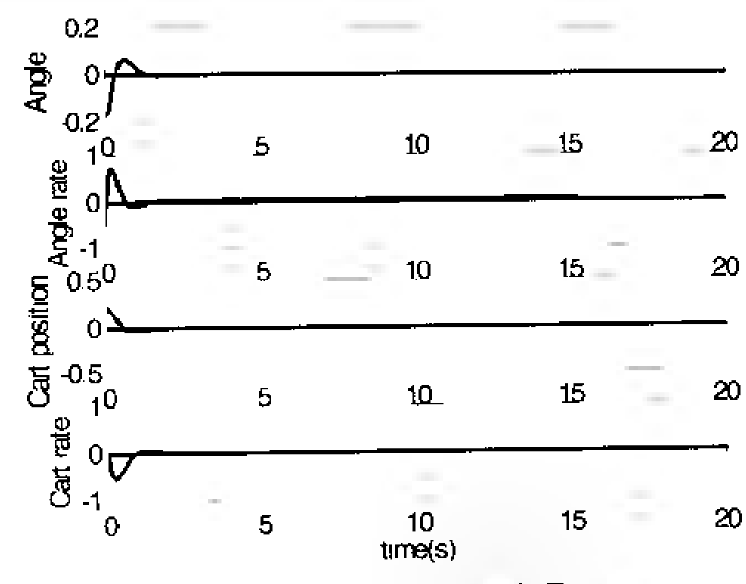


图 7-58 采用 PID 倒立摆响应结果（$S=2$）

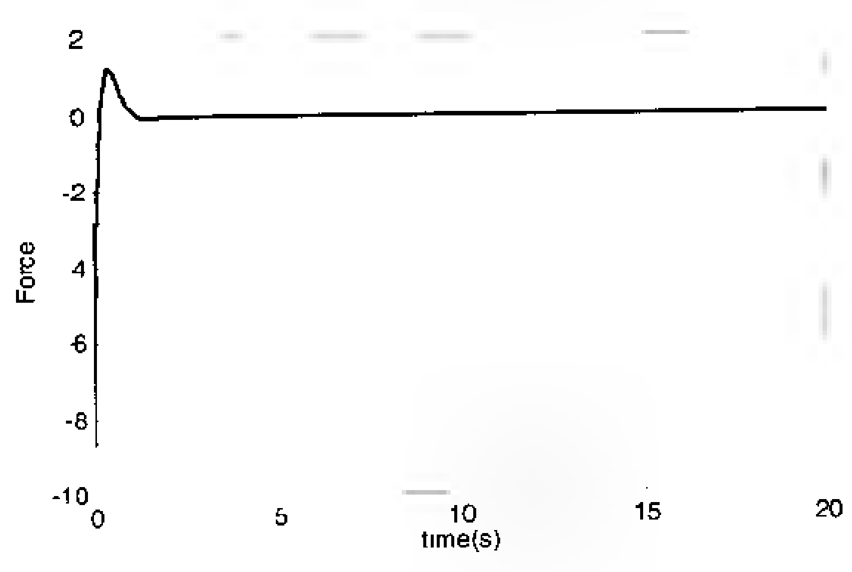


图 7-59 PID 控制器的输出

取 $S=2$，采用 LQR 控制，取 $Q=\begin{pmatrix} 100 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$，$R=0.10$，则由式（7.51）可得 LQR 控制器增益 $K=(-64.0799, 14.2285, -3.1623, 6.6632)$。

仿真程序由两部分组成：(1) 主程序 chap7_12.m，(2) 子程序 chap7_12f.m。

采用 LQR 时倒立摆响应结果及控制器输出如图 7-60 和图 7-61 所示，可见，采用 LQR

可实现单级倒立摆的最优控制。

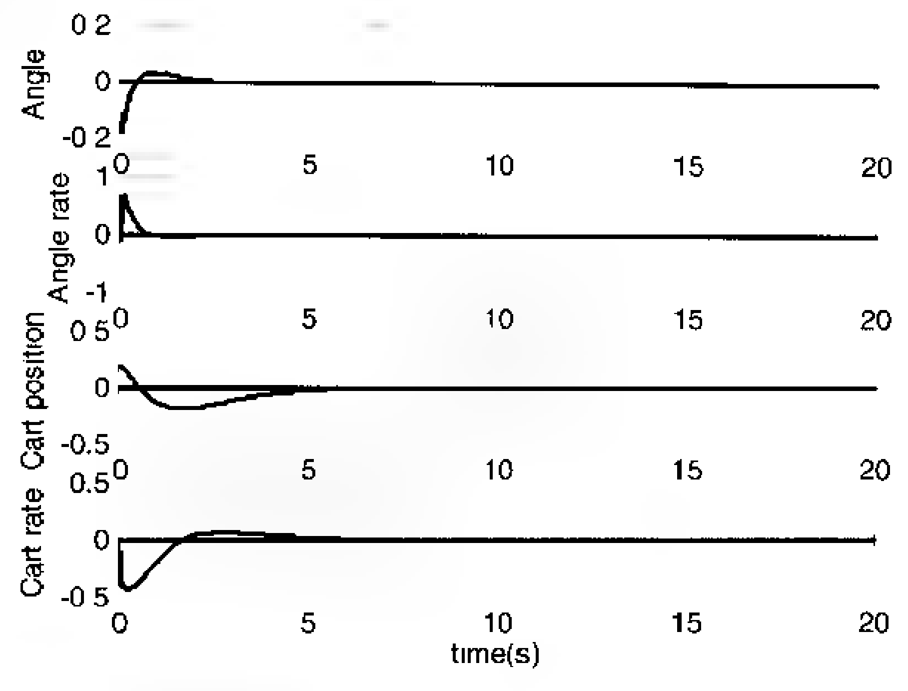


图 7-60 采用 LQR 倒立摆响应结果 S=1

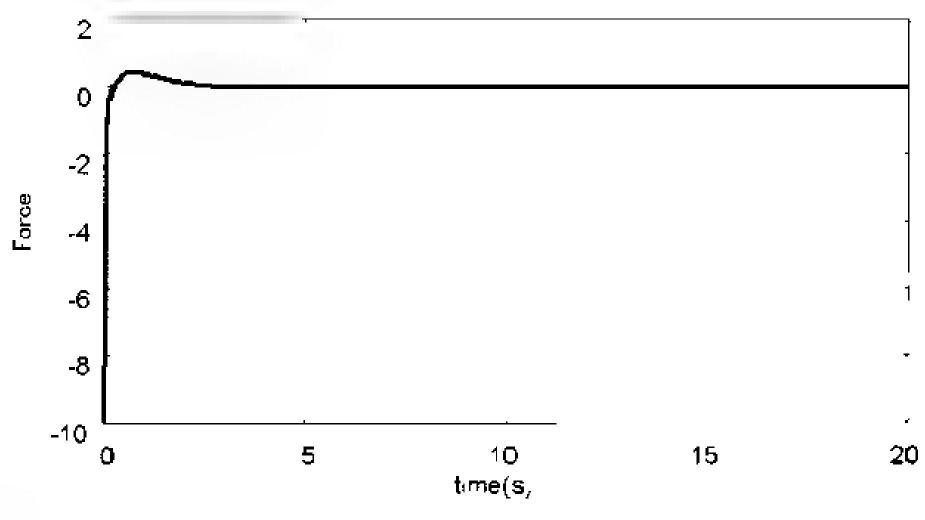


图 7-61 LQR 控制器的输出

主程序：chap7_12.m

```
%Single Link Inverted Pendulum Control
clear all;
close all;
global A B C D

%Single Link Inverted Pendulum Parameters
g=9.8;
M=1
m=0.1;
L=0.5
```

```
Fc  .000 ;
Fp 0 0000(2·

I 1 12*m*L^2;
l 1 2*L;
t1 m* M+m *g*l [ M+m *I+M*m*l^2];
t2  m^2*g*l^2/[ m+M)*I+M*m*l^2];
t3- m*l [ M+m)*I+M*m*l^2];
t4-(I+m*l^2)/[(m+M)*I+M*m*l^2];

A [0,1 0,0;
   t1,0,0,0;
   0,0,0,1;
   t2,0,0,0];
B [0,t3;0,t4];
C [1,0 0 0;
   0,0,1,0],
D-[0, ];

Q [100,0,0,0;   %100,10,1,1 express importance of theta,dtheta,x,dx
   0,10 0,0;
   0,0,1,0;
   0,0,0,1];
R [0 1];
K LQR A,B,Q,R ; %LQR Gain

e1_1 0;e2_1-0;e3 1 0;e4_1-0;
u_1- ;
xk [ 10 57.3,0,0.20,0];   %Initial state

ts-0.02,
for k 1:1:1000
time k) k*ts;
Tspan [0 ts];

para u_1];
[t,x] ode45( chap7_12f',Tspan,xk,[],para);
xk x length x ,:);

r1(k - 0;    %Pendulum Angle
r2 k ·0.0;   %Pendulum Angle Rate
r3(k  0 0;   %Car Position
```

```
r4(k, 0.0;    %Car Position Rate

x1 k) xk(1 ;
x2(k -xk(2 ;
x3(k)-xk,3 ;
x4(k, xk 4),

e1,k) r1(k, x1(k ;
e2(k) r2(k) x2 k ;
e3(k) r3(k) x3 k ;
e4(k)-r4(k  x4(k ;

S 1;
if S--1       %LQR
    u(k) K 1)*e1(k +K(2 *e2(k +K 3)*e3,k)(K 4)*e4 k);
elseif S -2  %PD
   de1(k, e1 k)-e1 1;
   u1(k)  50*e1(k  10*de1(k ;
   de2 k)-e2(k  e2 1;
   u2 k - 10*e2(k  10*de2(k);
   de3(k) e3(k)-e3 1;
   u3(k) -10*e3(k  10*de3(k);
   de4(k)-e4(k) e4 1;
   u4(k)  10*e4(k, 10*de4,k ;
   u(k, u1 k)+u2 k)+u3,k)+u4(k);
end

if u(k)> 10
   u(k,-10;
elseif u(k)<  10
    u(k) -10;
end
    e1_1 e1(k);
    e2_1-e2(k);
    e3_1-e3(k);
    e4_1 e4(k);
    u 1-u(k);
end
figure(1);
subplot(411 ;
plot(time,r1,'k',time,x1,'k  ;      %Pendulum Angle
xlabel('time(s) );ylabel('Angle',;
```

```
subplot(412);
plot(time,x2,'k',time,x2,'k');        %Pendulum Angle Rate
xlabel('time(s)');ylabel('Angle rate');
subplot(413);
plot(time,x3,'k',time,x3,'k');        %Car Position
xlabel('time(s)');ylabel('Cart position');
subplot(414);
plot(time,x4,'k',time,x4,'k');        %Car Position Rate
xlabel('time(s)');ylabel('Cart rate');
figure(2);
plot(time,u,'k');                     %Force F change
xlabel('time(s)');ylabel('Force');
```

子程序：chap7_12f.m。

```
function dx=dym(t,x,flag,para)
global A B C 1
u=para;
dx=zeros(4,1);

%State equation for one link inverted pendulum
dx=A*x+B*u;
```

7.8 吊车-双摆系统的控制

7.8.1 吊车-双摆系统的建模

实际的吊车要求将货物尽可能快地运送到目的地，并在移动过程中不能有大的晃动，这就要求吊车在移动过程中保持上下摆角平稳而且小车本身又要达到指定的位置，这些要求可通过电机的控制来实现。吊车-双摆系统如图 7-62 所示。

图中所标参数：M 为吊车质量；m_1 为上摆质量；m_2 为下摆质量；x 为小车位置；β 为下摆角；α 为上摆角；l_1 为上摆杆长度；l_2 为下摆杆长度；F 为拉小车的力。

通过受力分析，进行线性化处理，令 $x=[x\ \ \dot{x}\ \ \alpha\ \ \dot{\alpha}\ \ \beta\ \ \dot{\beta}]$，可得到关于小车、上摆角、下摆角的状态方程：

$$\dot{x}=\begin{bmatrix}0 & 1 & 0 & 0 & 0 & 0\\ 0 & \dfrac{-\mathrm{KeKt}}{\mathrm{Ra}(\mathrm{Mr}^2+J)} & \dfrac{-(m_1+m_2)gr^2}{\mathrm{Mr}^2+J} & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & 0 & 0\\ 0 & \dfrac{-\mathrm{KeKt}}{\mathrm{Ra}(\mathrm{Mr}^2+J)} & \dfrac{-[(M+m_1+m_2)r^2+J]g}{(\mathrm{Mr}^2+J)l} & 0 & \dfrac{m_2g}{m_1l} & 0\\ 0 & 0 & 0 & 0 & 0 & 1\\ 0 & \dfrac{\mathrm{KeKt}}{\mathrm{Ra}(\mathrm{Mr}^2+J)} & \dfrac{[(M+m_1+m_2)r^2+J]g}{(\mathrm{Mr}^2+J)l_1} & 0 & \dfrac{-(m_2l_2+lm_1+lm_2)g}{m_1l\,l_2} & 0\end{bmatrix}x+\begin{bmatrix}0\\ \dfrac{r\mathrm{Kt}}{\mathrm{Ra}(\mathrm{Mr}^2+J)}\\ 0\\ \dfrac{r\mathrm{Kt}}{\mathrm{Ra}(\mathrm{Mr}^2+J)l_1}\\ 0\\ -\dfrac{r\mathrm{Kt}}{\mathrm{Ra}(\mathrm{Mr}^2+J)l_1}\end{bmatrix}u$$



$$y=\begin{bmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&1&0&0&0\\0&0&0&1&0&0\\0&0&0&0&1&0\\0&0&0&0&0&1\end{bmatrix}x \tag{7.53}$$

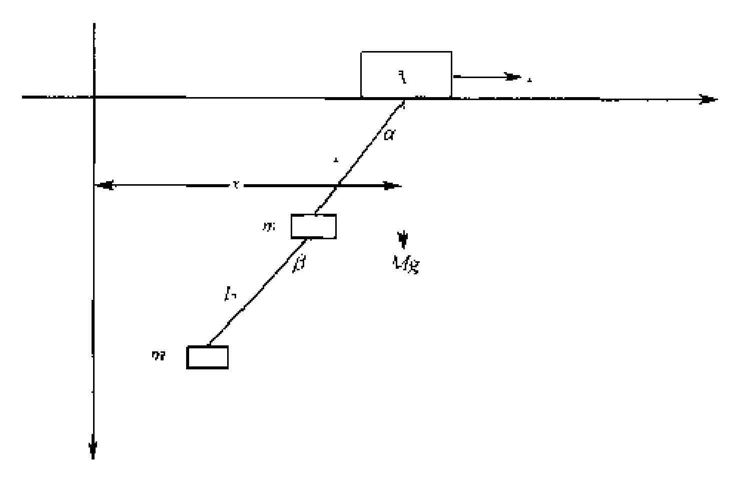


图 7-62 吊车-双摆系统

7.8.2 吊车-双摆系统的仿真

取某吊车-双摆参数为[28]：电机负载 $J=1\times10^{-4}\text{kg}\cdot\text{m}^2$，反电势系数 Ke=0.4758V.s，电枢电阻 Ra=13.5Ω，力矩系数 Kt=0.0491kg·m/A，传送轮半径 $r=0.02276\text{m}$，$m_1=0.3\text{kg}$，$m_2=0.5\text{kg}$，$M=0.4\text{kg}$，$l_1=0.205\text{m}$，$l_2=0.156\text{m}$，将上述参数带入式(7.53)，可得实际系统的状态方程：

$$\dot{x}=Ax+Bu$$
$$y=Cx+D$$

式中，

$$A=\begin{bmatrix}0&1&0&0&0&0\\0&58.1558&13.3099&0&0&0\\0&0&0&1&0&0\\0&-73.7445&-112.7311&0&79.6748&0\\0&0&0&0&0&1\\0&73.7445&112.7311&0&247.1962&0\end{bmatrix}$$

$$B=[0\ \ 9.48888\ \ 0\ \ 46.275\ \ 0\ \ -46.275]^{\mathrm T},\quad C=\begin{bmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&1&0&0&0\\0&0&0&1&0&0\\0&0&0&0&1&0\\0&0&0&0&0&1\end{bmatrix},\quad D=[0\ 0\ 0\ 0\ 0\ 0]^{\mathrm T}$$

连续系统的控制器采用连续系统的 S 函数来实现。在 S 函数中，只采用初始化和输出函数，即 mdlInitializeSizes 函数和 mdlOutputs 函数。在初始化中采用 Sizes 结构，选择 1 个输

出，7 个输入，在 7 个输入中，前 6 个输入为[illegible]，第 7 个输入为小车的目标位置，在输出中[illegible]，其中 S=1 时为 LQR 控制，S=2 时为 PID 控制，S 函数嵌入在 Simulink 程序中。

[illegible]为：$x(0)=0$，$\dot{x}(0)=0$，$\alpha(0)=1.2$，$\dot{\alpha}(0)=0$，$\beta(0)=-1.2$，$\dot{\beta}(0)=0$。期望状态为：$x(0)=0.10$，$\dot{x}(0)=0$，$\alpha(0)=0$，$\dot{\alpha}(0)=0$，$\beta(0)=0$，$\dot{\beta}(0)=0$。[illegible]

在 Simulink [illegible]，[illegible]$\frac{180}{\pi}$，[illegible]仿真程序由[illegible]组成：(1) [illegible] chap7_13.m；(2) Simulink [illegible] chap7_13sim.mdl；(3) S 函数程序 chap7_13s.m。

取 $S=1$，采用 LQR 控制，取 $Q=\begin{pmatrix}1000&0&0&0&0&0\\0&1&0&0&0&0\\0&0&100&0&0&0\\0&0&0&1&0&0\\0&0&0&0&100&0\\0&0&0&0&0&1\end{pmatrix}$，小车目标位置为 $R=0.10$，

则由式（7.51）可得 LQR [illegible] $K=(100, 1.0370, 32.6938, 5.0696, -48.7526, 0.6557)$。小车[illegible]图 7-63～图 7-68 [illegible]，[illegible] LQR 可实现[illegible]双摆的最优控制。

取 $S=2$，采用 P[illegible]，[illegible]的仿真为例，其响应结果[illegible]图 7-69 [illegible]。可见，[illegible] PID [illegible]双摆[illegible]。

[illegible]双摆系统[illegible]，说明了 PID [illegible]。

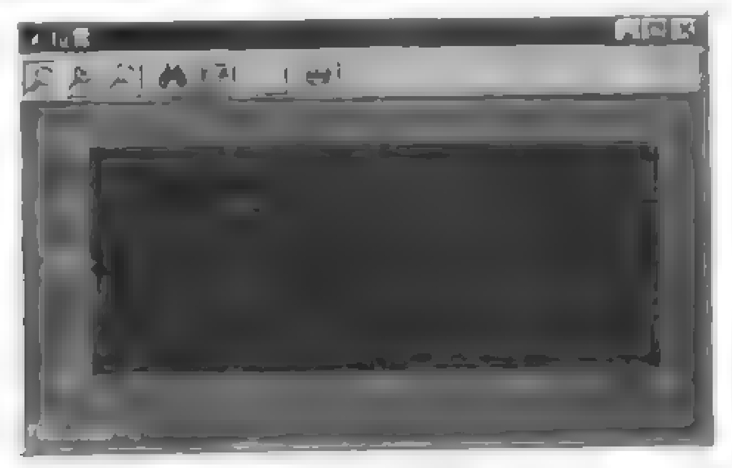


图 7-63　小车位置响应

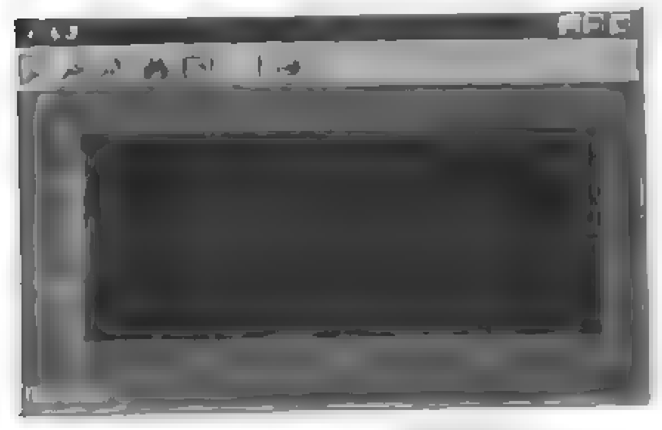


图 7-64　小车[illegible]响应

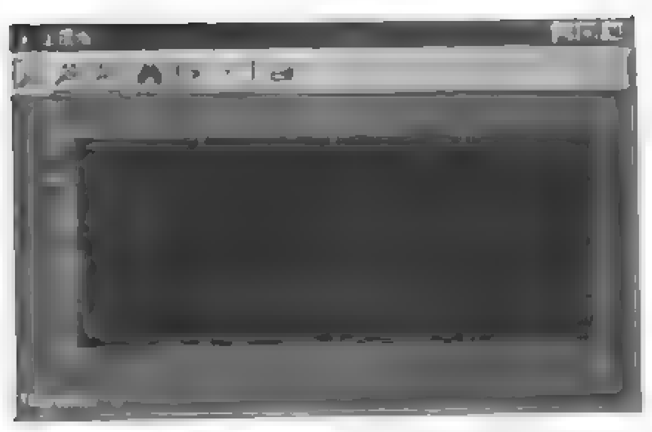


图7-65　上摆角角度响应

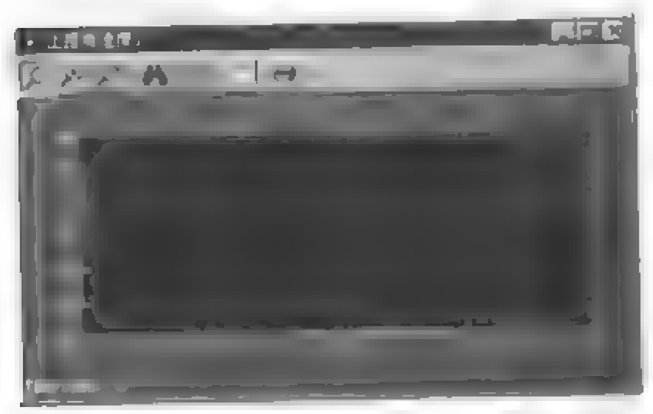


图7-66　上摆角角速度响应



图7-67　下摆角角度响应

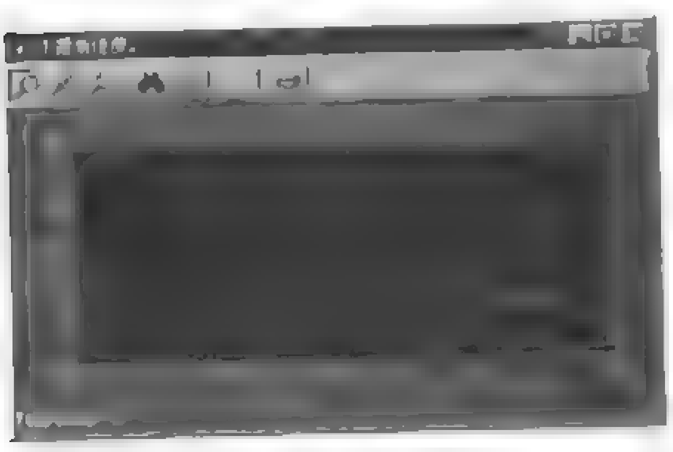


图7-68　下摆角角速度响应

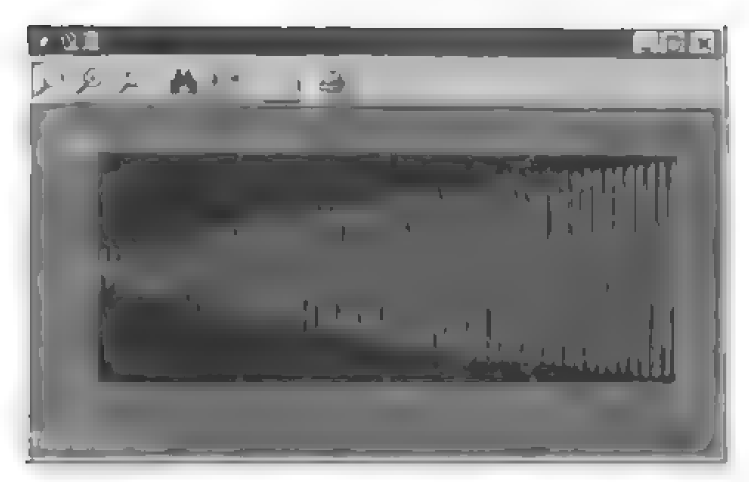


图7-49

主程序：chap7_13.m

Simulink 程序：chap7_13sim.mdl，如图 7.70 所示。

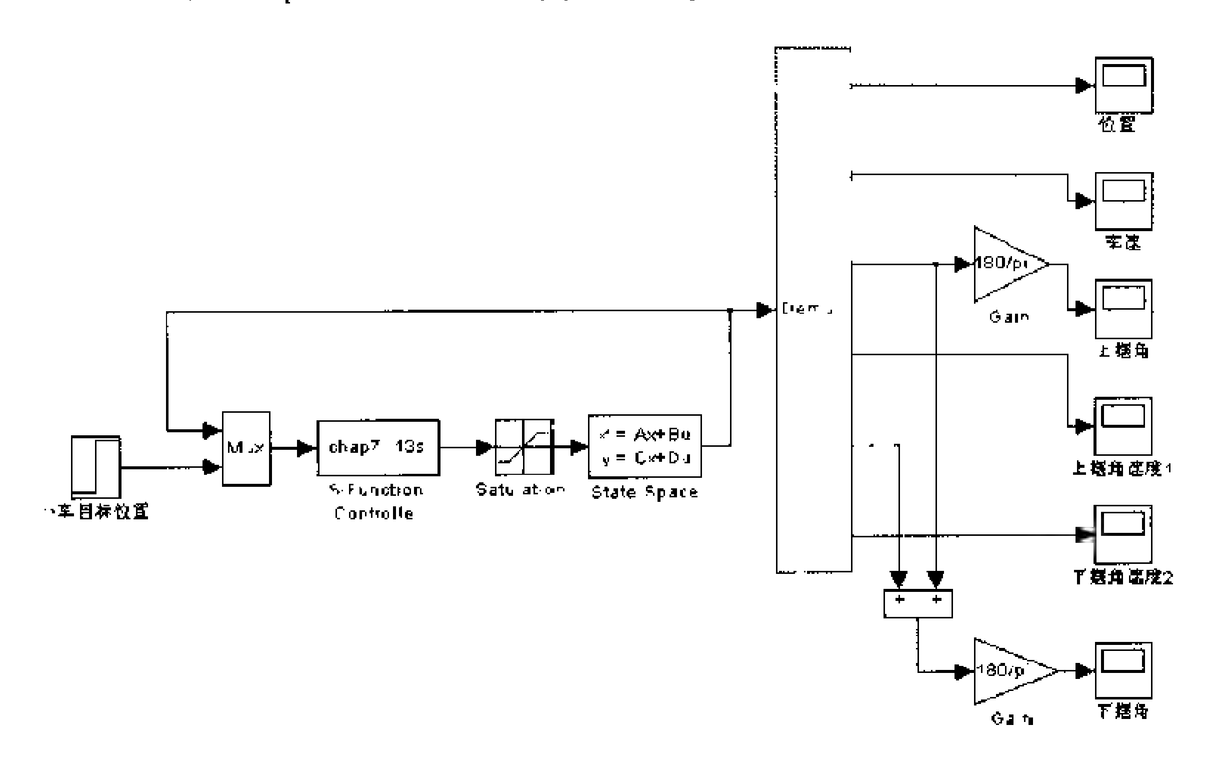


图 7-70 小车双摆的 Simulink 仿真模块

S 函数程序：chap7_13s.m。

```
% S-function for continuous state equation
function [sys,x0,str,ts]=s_function(t,x,u,flag)

switch flag,
%Initialization
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
% Outputs
  case 3,
    sys=mdlOutputs(t,x,u);
%Unhandled flags
  case {1, 2, 4, 9 }
    sys=[];
%Unexpected flags
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end

%mdlInitializeSizes
function [sys,x0,str,ts]=mdlInitializeSizes
sizes=simsizes;
```

```
    sizes.NumContStates  = 0;
    sizes.NumDiscStates  = 0;
    sizes.NumOutputs     = 1;
    sizes.NumInputs      = 7;
    sizes.DirFeedthrough = 0;
    sizes.NumSampleTimes = 0;

    sys=simsizes(sizes);
    x0=[];
    str=[];
    ts=[];

    %mdlOutputs
    function sys=mdlOutputs(t,x,u)
    global K
    S=1;
    if S==1
       sys=-K*[u(1)-u(7),u(2),u(3),u(4),u(5),u(6)]';    %LQR
    elseif S==2
sys=50*(u(7)-u(1))+10*(0-u(2))+10*(0-u(3))+10*(0-u(4))-10*(0-u(5))+10*(0-u(6))
);  %PID
    end
```

7.9 基于 Anti-windup 的 PID 控制

7.9.1 Anti-windup 的基本原理

任何实际的控制系统都包含饱和非线性特性。控制器的 Windup 问题一般被认为是当控制器输出 u 和输入 $\hat{u}$ 之间存在非线性特性 $N(u)$ 时，产生于控制器 PI/PID 积分部分的一种不良现象。

饱和特性对实际系统的影响十分严重。由于在系统调试过程中大都以小信号作为系统的调试信号，所以造成设计者对饱和特性非线性的认识不足而忽略了它的存在。在实际过程中，当有大信号输入或其他情况使控制系统进入饱和状态时，系统的性能会产生较大的降低，不能满足性能的要求。因此，引入适当的补偿环节，使控制系统在出现饱和现象时仍能达到比较满意的性能指标的 Anti-windup 设计技术成为进行具有饱和特性的控制系统设计的基本思路。

在参考文献[29]中针对积分 Windup 现象，提出了一种变结构 PID 控制器，其结构如图 7-71 所示。

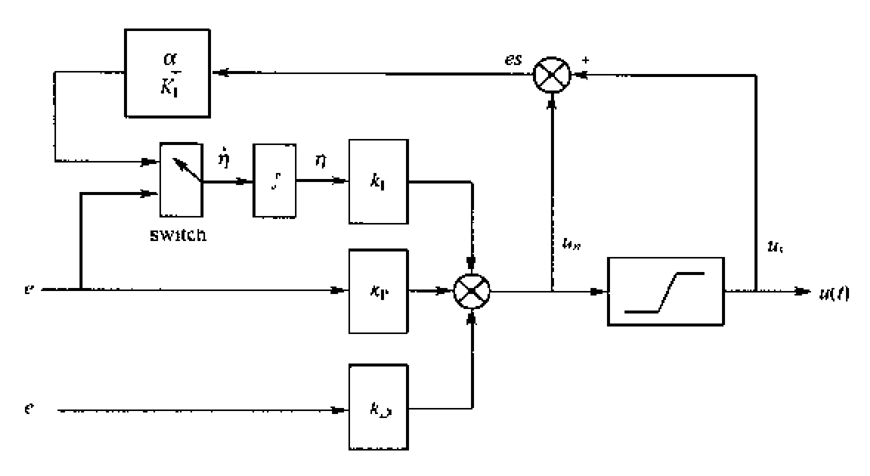


图 7-71　基于 Anti windup 的 PID 控制器结构

当控制器输出信号饱和时，饱和误差项为$u_n - u_s$，采用系数η实现积分项的自适应调整。η的自适应变化律为：

$$\dot{\eta} = \begin{cases} -\alpha(u_n - u_s)/K_1, & u_n \neq u_s, e(u_n - \bar{u}) > 0, \bar{u} = (u_{\min} + u_{\max})/2 \\ e & u_n = u_s \end{cases} \tag{7.54}$$

式中，$\alpha > 0$。

7.9.2 仿真程序及分析

仿真实例

设被控对象为：

$$\ddot{\theta} + 25\dot{\theta} = 133u$$

指令为一个大的阶跃信号 1500。分别采用控制器式（7.54）和传统 PID 进行仿真。当M=1 时为 Anti-windup 的 PID 控制，当M=2 时为传统 PID 控制。仿真方法分为以下两种。

仿真方法一

针对连续系统，采用离散控制器进行仿真，采样时间为 1s，取$\alpha = 1.0$，$k_p = 0.10$，$k_i = 0.01$，$k_d = 0.01$，控制器输出信号的范围为 0～10。

仿真结果如图 7-72～图 7-75 所示。

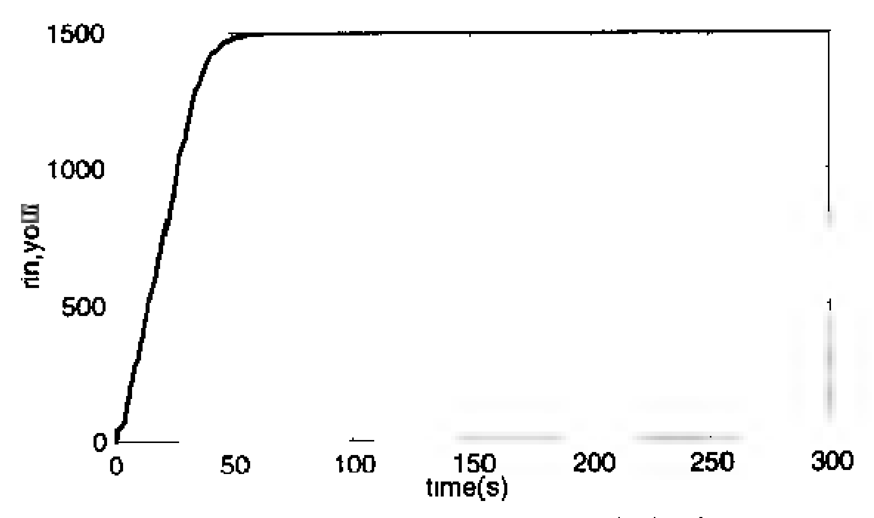


图 7-72　基于 Anti windup 方法的 PID 阶跃响应(M=1)

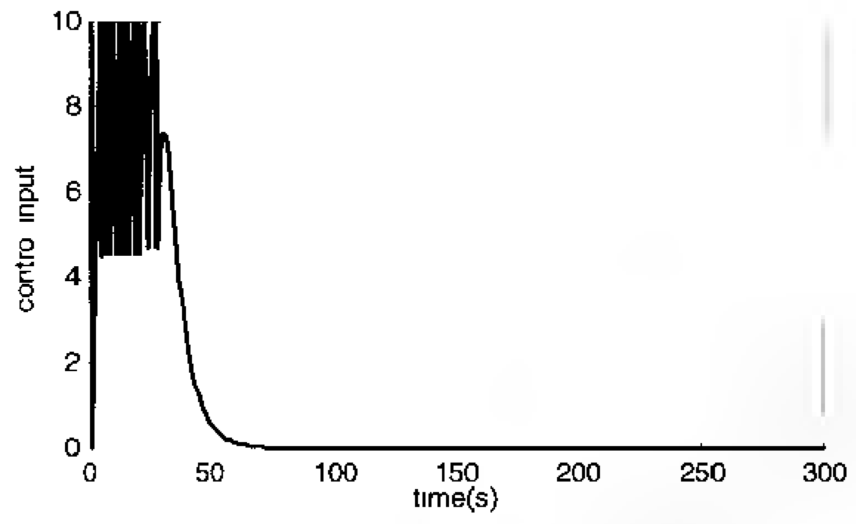


图 7-73 基于 Anti windup 方法的控制器输出

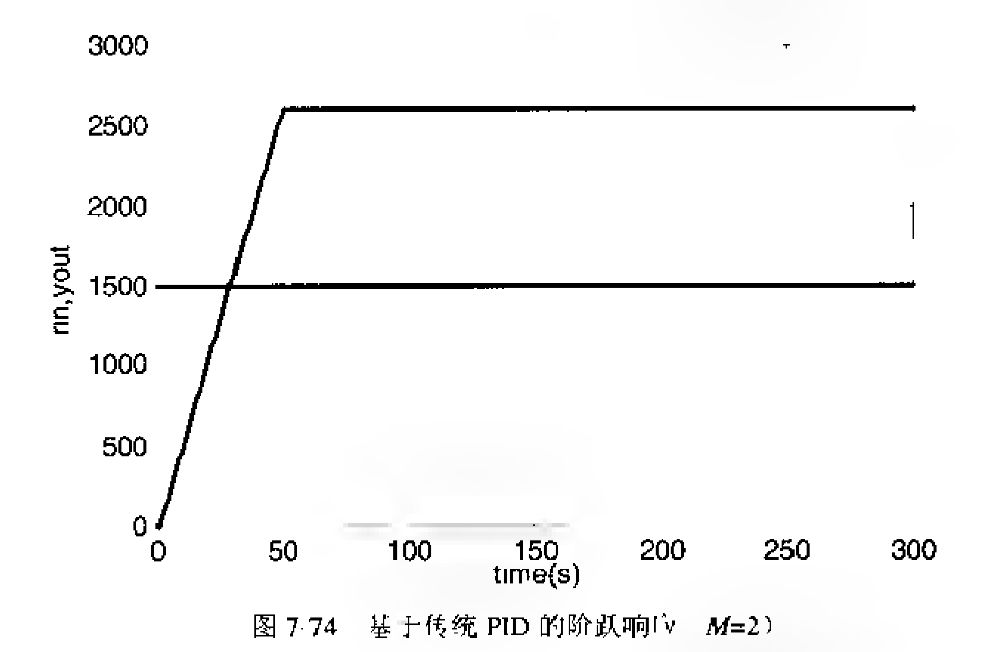


图 7-74 基于传统 PID 的阶跃响应（$M=2$）

10
8
6
4
2
0
control input
0
50
100
150
200
250
300
time(s)

图 7-75 传统 PID 方法的控制器输出

仿真程序：chap7_14.m

```
%Discrete PID control with A[illegible]
clear [illegible]
cl[illegible]e all;

xk=zer[illegible]2,1[illegible];
e[illegible];
e1[illegible];
[illegible]

alfa=1.[illegible];

kp=[illegible];
ki=[illegible];
kd=[illegible];

umin[illegible]
umax=[illegible];

ts=[illegible];
for k=1:[illegible]0
time(k)=k*ts;

rin(k)=[illegible]50[illegible]

p[illegible];
t[illegible]para[illegible]t[illegible];
[illegible]ode45('chap7_[illegible]t[illegible]xk,[illegible]para[illegible]
xk=xx(length(xx)[illegible]);
y[illegible](k)=xk[illegible];

e(k)=rin(k)-yo[illegible](k);
de(k)=(e(k)-e_1)[illegible]

un(k)=kp*e(k)+ki*[illegible]+kd*de(k);

[illegible](k)=[illegible](k);
if [illegible](k)>=umax
   [illegible](k)=umax;
end
if [illegible](k)<=umin
```

```
  us(k)=umin;
end

es(k)=us(k)-un(k);

M=1;
switch M
case 1        %VSPID
    ua(k)=(umax+umin)/2;
    if un(k)~=us(k)&e(k)*(un(k)-ua(k))>0
       ef(k)=-alfa*(un(k)-us(k))/ki;
    else
       ef(k)=e(k);
    end
  end
case 2        %No Anti-windup
  ef(k)=e(k);
end
ei=ei+ef(k)*ts;

u_1=us(k);
e_1=e(k);
end
figure(1);
plot(time,rin,'r',time,yout,'b');
xlabel('time(s)'),ylabel('rin,yout');
figure(2);
plot(time,us,'r');
xlabel('time(s)'),ylabel('control input');
```

被控对象子程序：chap7_14f.m。

```
function dx=PlantModel(t,x,flag,para)
dx=zeros(2,1);

u=para;

dx(1)=x(2);
dx(2)=-25*x(2)+133*u;
```

仿真方法二

针对连续系统，采用 Simulink 方式进行连续控制器的仿真，取$\alpha=1.0$，$k_p=50$，$k_i=10$，$k_d=1$，控制器输出信号的范围为0～10。仿真结果如图7-76～图7-79所示。

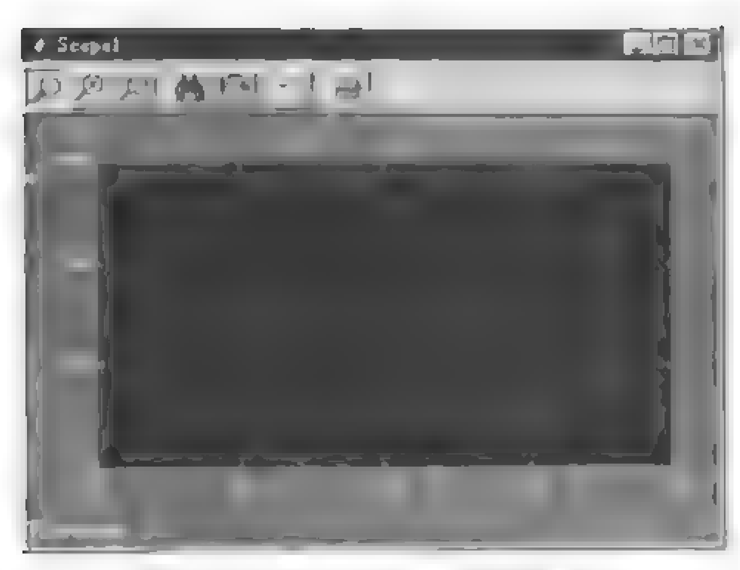
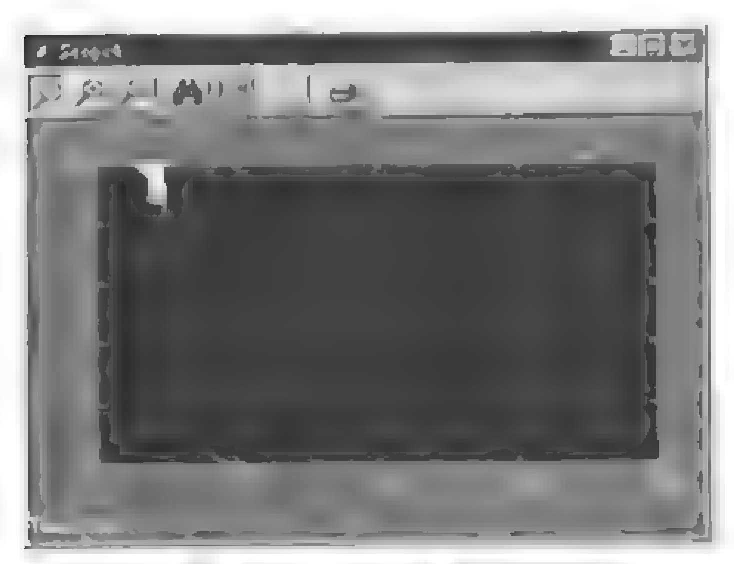
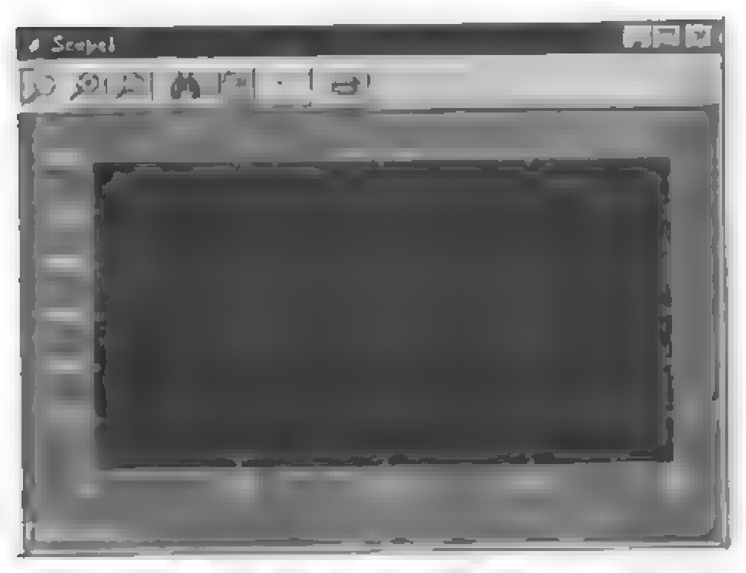


图 7-76 [illegible] Anti-windup [illegible]



[illegible] Anti-windup [illegible]



[illegible] 传统 PID 的 [illegible]

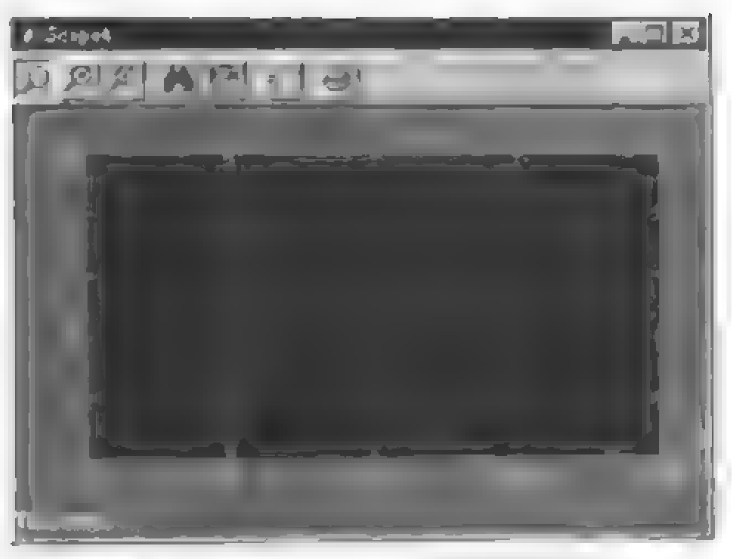


图 7-79 传统 PID 方法仿真结果

仿真程序：Simulink 主程序：chap7_15.mdl，如图 7-80 所示。

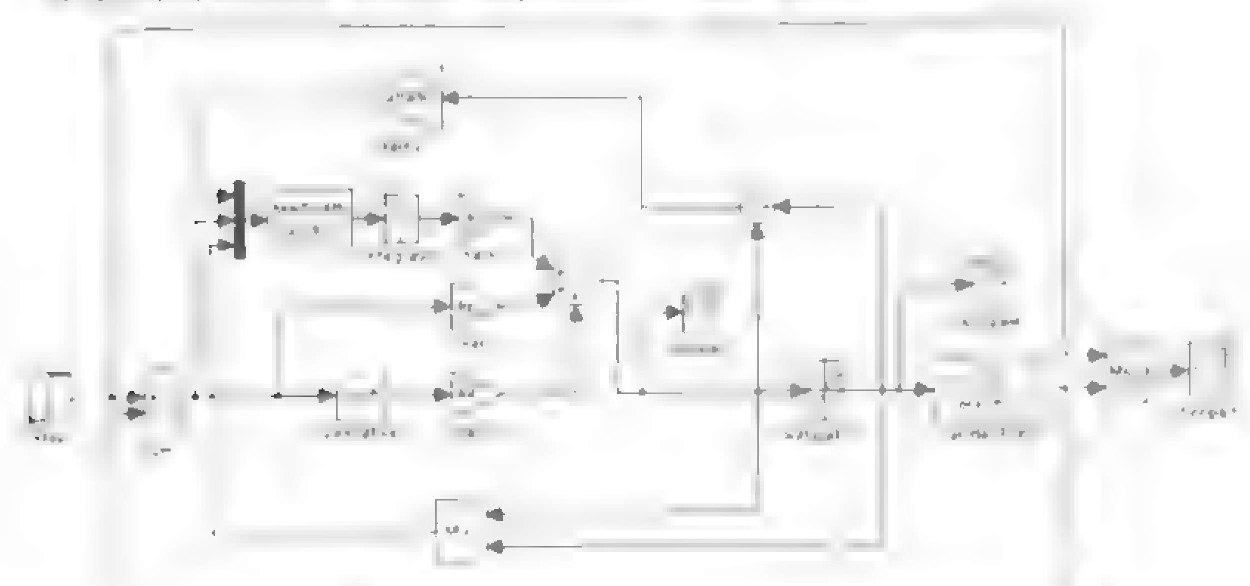


图 7-80 Anti-windup 的 Simulink 仿真程序

初始化程序：chap7_15f.m

```
umin=0;
umax=10;

ua=(umin+umax)/2;
```

M 函数子程序：chap7_15m.m

```
function [u]=pid_awlf1(u1,u2,u3,u4)

e=u2;
un=u1;
xx=u4;

M=1;
switch M
case 1              %PID
    u=e;
case 2              %anti windup PID
    umin=0;
    umax=10;
    ua=(umin+umax)/2;
    if abs(un)>=umax&e*(un-ua)>0
        u=u1;
    else
        u=e;
    end
end
```

7.10 基于 PD 增益自适应调节的模型参考自适应控制

7.10.1 控制器的设计

设被控对象为：

$$\ddot{\theta}+\alpha_1\dot{\theta}+\alpha_0\theta=\beta_0 u \tag{7.55}$$

式中，θ 为系统输出转角，u 为控制输入，α_0、α_1 为非负的实数，β_0 为正实数。

定义参考模型为：

$$\ddot{\theta}_m+a_1\dot{\theta}_m+a_0\theta_m=br \tag{7.56}$$

式中，θ_m 为模型输出，r 为系统指令输入，a_0、a_1、b 为正实数。

定义误差信号为：

$$e=\theta_m-\theta \tag{7.57}$$

由式（7.55）～式（7.57），得到误差动态方程如下：

$$\ddot{e}+a_1\dot{e}+a_0 e=br-\beta_0 u+(\alpha_1-a_1)\dot{\theta}+(\alpha_0-a_0)\theta \tag{7.58}$$

定义$\boldsymbol{\varepsilon} = [e \quad \dot{e}]^{\mathrm{T}}$，则得到误差状态方程如下：

$$\dot{\boldsymbol{\varepsilon}} = \boldsymbol{A}\boldsymbol{\varepsilon} - \begin{bmatrix} 0 \\ \beta \end{bmatrix} u + \begin{bmatrix} 0 \\ \Delta \end{bmatrix} \tag{7.59}$$

式中，$\Delta = br + (\alpha_0 - a_0)\theta + (\alpha_1 - a_1)\dot{\theta}$，$\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ -a_0 & -a_1 \end{bmatrix}$

由于矩阵$\boldsymbol{A}$的特征值具有负实部，所以式（7.59）表示的模型是稳定的，则存在正定矩阵$\boldsymbol{P}$和$\boldsymbol{Q}$，使得下式成立：

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{P} + \boldsymbol{P}\boldsymbol{A} = -\boldsymbol{Q} \tag{7.60}$$

在参考文献[43]中，以PD形式定义控制项$\hat{e}$为：

$$\hat{e} = p_1 e + p_2 \dot{e} \tag{7.61}$$

式中，$[p_1 \quad p_2] = [0 \quad 1]\boldsymbol{P}$

在参考文献[44]中，设计自适应控制律为：

$$u = k_0 r + k_1 \theta + k_2 \dot{\theta} \tag{7.62}$$

$$\dot{k}_0 = \lambda_0 \hat{e} r \tag{7.63}$$

$$\dot{k}_1 = \lambda_1 \hat{e} \theta \tag{7.64}$$

$$\dot{k}_2 = \lambda_2 \hat{e} \dot{\theta} \tag{7.65}$$

7.10.2 稳定性分析

设计Lyapunov函数为：

$$V = \frac{1}{2}\boldsymbol{\varepsilon}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{\varepsilon} + \frac{1}{2\lambda_0\beta_0}(b - \beta_0 k_0)^2 + \frac{1}{2\lambda_1\beta_0}(\alpha_0 - a_0 - \beta_0 k_1)^2 + \frac{1}{2\lambda_2\beta_0}(\alpha_1 - a_1 - \beta_0 k_2)^2 \tag{7.66}$$

对V求导数，由于

$$\left(\frac{1}{2}\boldsymbol{\varepsilon}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{\varepsilon}\right)' = \frac{1}{2}\boldsymbol{\varepsilon}^{\mathrm{T}}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{P} + \boldsymbol{P}\boldsymbol{A})\boldsymbol{\varepsilon} + \boldsymbol{\varepsilon}^{\mathrm{T}}\boldsymbol{P}\begin{bmatrix} 0 \\ 1 \end{bmatrix}(\Delta - \beta_0 u)$$

考虑到式（7.60），且有

$$\boldsymbol{\varepsilon}^{\mathrm{T}}\boldsymbol{P}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = [0 \quad 1]\boldsymbol{P}\boldsymbol{\varepsilon} = \hat{e}$$

则

$$\dot{V} = -\frac{1}{2}\boldsymbol{\varepsilon}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{\varepsilon} + \hat{e}(\Delta - \beta_0 u) - \frac{\dot{k}_0}{\lambda_0}(b - \beta_0 k_0) - \frac{\dot{k}_1}{\lambda_1}(\alpha_0 - a_0 - \beta_0 k_1) - \frac{\dot{k}_2}{\lambda_2}(\alpha_1 - a_1 - \beta_0 k_2)$$

将Δ和u代入上式，得：

$$\hat{e}(\Delta - \beta_0 u) = \hat{e}r(b - \beta_0 k_0) + \hat{e}\theta(\alpha_0 - a_0 - \beta_0 k_1) + \hat{e}\dot{\theta}(\alpha_1 - a_1 - \beta_0 k_2)$$

则

$$\dot{V} = -\frac{1}{2}\boldsymbol{\varepsilon}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{\varepsilon} + \left(\hat{e}r - \frac{\dot{k}_0}{\lambda_0}\right)(b - \beta_0 k_0) + \left(\hat{e}\theta - \frac{\dot{k}_1}{\lambda_1}\right)(\alpha_0 - a_0 - \beta_0 k_1) + \left(\hat{e}\dot{\theta} - \frac{\dot{k}_2}{\lambda_2}\right)(\alpha_1 - a_1 - \beta_0 k_2)$$

将式（7.63）～式（7.65）代入，得：

$$\dot{V} = -\frac{1}{2}\boldsymbol{\varepsilon}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{\varepsilon} \leqslant 0$$

7.10.3 仿真程序及分析

仿真实例

设被控制对象为：

$$\ddot{\theta}+20\dot{\theta}+25\theta=133u$$

参考模型为：

$$\ddot{\theta}_m+20\dot{\theta}_m+30\theta_m=50r$$

指令信号分两种，当 $S=1$ 时为方波，当 $S=2$ 时为正弦。取 $S=1$，即指令信号为方波，控制器参数取 $\lambda_0=\lambda_1=\lambda_2=200$，正定矩阵取 $Q=\begin{bmatrix}20 & 10\\ 10 & 20\end{bmatrix}$。

仿真过程中，被控对象初始状态取为[0，0]，参考模型初始状态取为[0，0]，自适应参数 k_0、k_1、k_2 的初始状态取为[0，0，0]。

仿真结果如图 7-81～图 7-84 所示，图 7-81 和图 7-82 为参考模型位置跟踪及跟踪误差，图 7-83 为控制器输出，图 7-84 为控制器参数 k_0、k_1、k_2 的自适应变化过程。

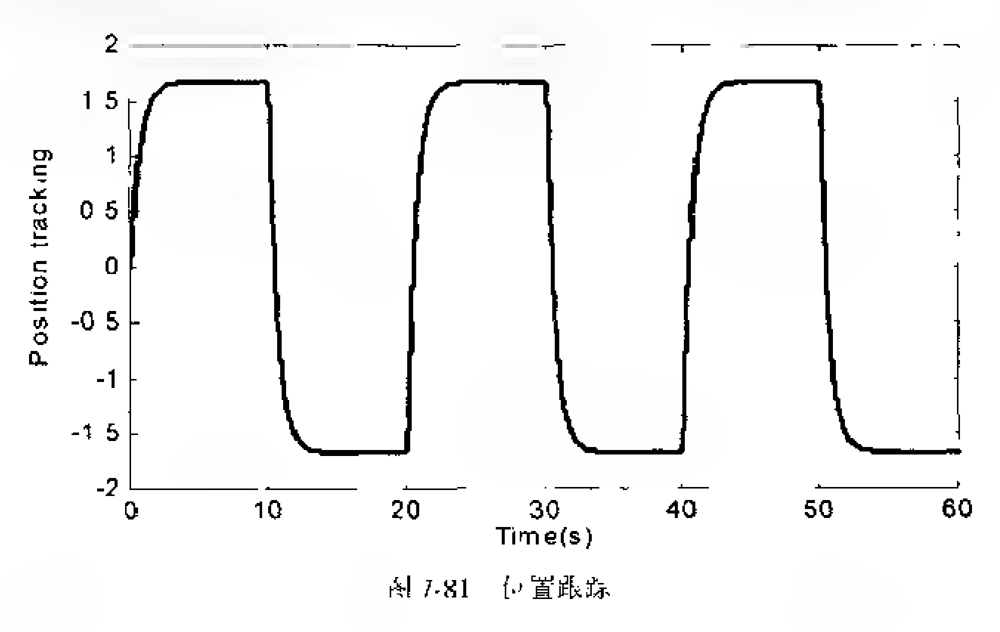


图 7-81 位置跟踪

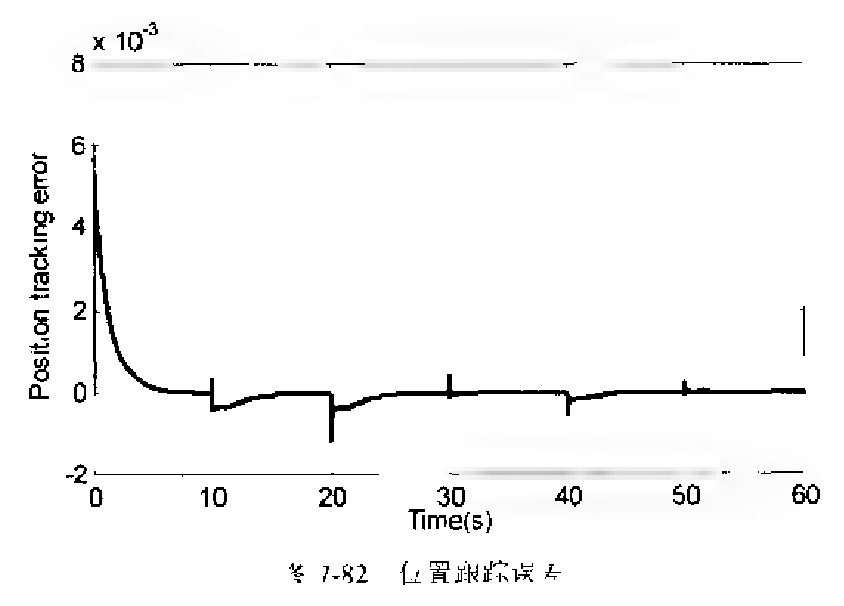


图 7-82 位置跟踪误差

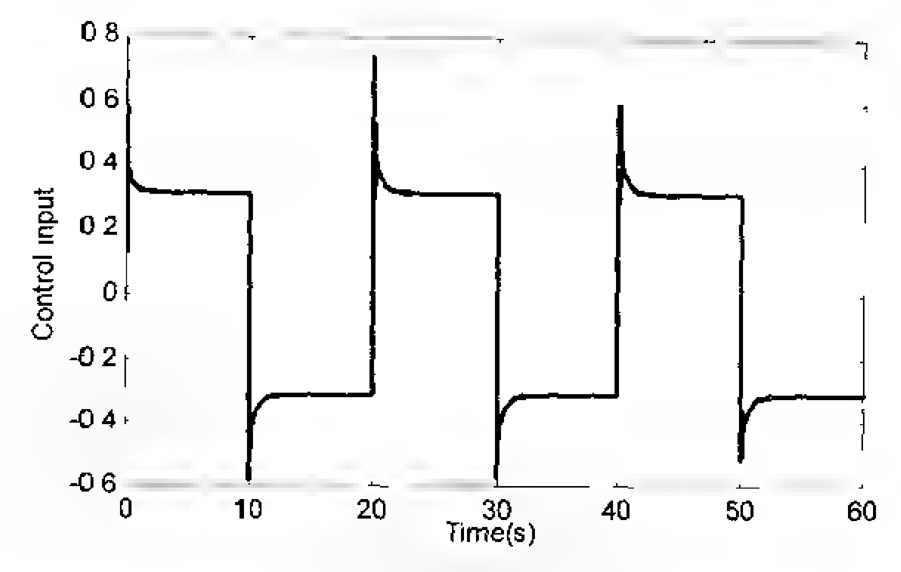


图 7-83　控制器输出信号

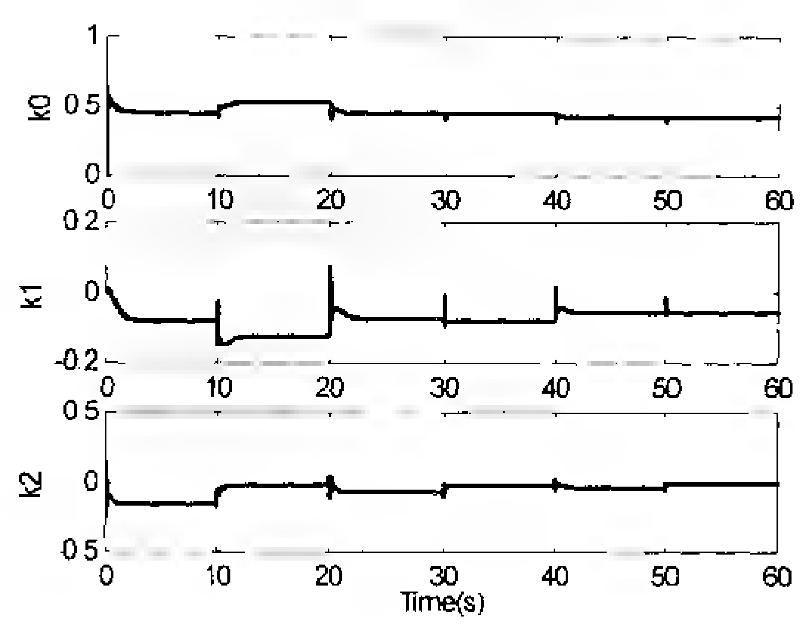


图 7-84　k_0,k_1,k_2的变化过程

主程序：chap7_16.m

```
%Adaptive Robust Control Based on PD Term
clear all;
close all;
global

ts=0.001;
TimeSet=[0:ts:60];

a0=10;
a=20;
b=50;
```

```
A= ... a0, a1 ;
eig(A=
%Q 20,0;0,20]·
... 2 ...10 20]·

P ...(A',...
p12 P ... 2 ;
p22 P ...2,2 ·
para ... a1 ...0,b,r11,r22];

 t yout  ode45 ... ifeq ,TimeSt , 0 0 ... ],[],para ,
k0 yout :,5 ;
k1 yout  ,6 ;
k2 yout  ,7 ;

switch S
case 1
   r  ...*sign(sin ...5*t*2*pi  ;    %Square Signal
case 2
   r 1...*sin ...0*t*2*pi ,         %Sin Signal
end
u k0.*r+k1.*yout :,3 +k2.*yout :,4 ·

figure 1,·
plot t yout  ,1  'k ,t,yout ...,3   k' ;
xlabel  'Time s   ,ylabel  Position tracking  ,

figure 2);
 lo  t yout  ,1  yout :,3   k');
xlabel 'Time s   ;ylabel  Position tracking error  ;

figure 3)·
plot(t,u, k  ;
xlabel 'Time ...  ;ylabel 'Control input  ;

figure 4);
subplot 3,1,1 ;
plo  t,k0, k' ;
xlabel 'Time s   ;ylabel  k0' ;
subplot 3,...,2
plot t,k1  k' ;
```

```
xlabe   Time s   ;ylab    k    ;
subplot(3,1,3);
plot    k2   k )
xlabel  Time s  ;ylabel  k2   ;
```

子程序：chap7_16eq.m

```
fun         1, DynamicModel  t  y   para
globa  S
dy zeros    ;

S  ;
switch S
case
   r 1.  *sign          *r*  *p    ;     %Square signal
case 2
   r    0*sin 1.0*t*  *p   ;            %Sin signal
end

 1 para    ;
a  para 2  ;
b para    ;
p12=para(4 ;
p22 para(5 ;

e y    y(
de y 2   y 4
eF p  *e+p 2*de;

k0 y 5  ;
k  y 6  ;
k2 y    ;
u k0*r+k1*y  1 +k2*y 4 ;

dy(1  y 2 ;
dy 2   b*r  a *y 2   a *y(1  ;

dy 3  y   ;
dy 4    25*y      *y 4 +1  *u;

dy 5   200*eF*r;         %k0
dy 6   200*eF*y   ;      %k1
dy     200*eF*y 4 ;      %k2
```

第8章 灰色PID控制

8.1 灰色控制原理

部分信息已知、部分信息未知的系统为灰色系统。灰色预测是用灰色模型GM(M,N)进行的定量预测，灰色控制是指对本征特性灰色系统的控制，或系统中含灰参数的控制，或用GM(M,N)构成的预测控制。

8.1.1 生成数列

1. 累加生成（Accumulated Generating Operation，AGO）

如果有一串原始数列，第一个维持不变，第二个数据是原始的第一个加第二个数据，第三个数据是原始的第一个、第二个与第三个相加……这样得到的新数列，称为累加生成数列，这种处理方式称为累加生成。

基本公式：

$$x^{(1)}(k)=\sum_{i=1}^{k}x^{(0)}(i) \tag{8.1}$$

$$x^{(1)}(k)=x^{(1)}(k-1)+x^{(0)}(k) \tag{8.2}$$

累加生成是使任意非负序列、摆动的与非摆动的数据转化为非减的、递增的数据。

没有规律的原始数据，经累加生成后，如果能得到较强的规律，并且接近某一函数，则该函数称为生成函数。生成函数就是一种模型，称为生成模型。通过累加获得的模型称为累加生成模型。

2. 累减生成

累减生成是指将原始数列前后两个数据相减所得的数据。累减生成是累加生成的逆运算，简记为IAGO(Inverse Accumulated Generating Operation)。累减运算可将累加生成还原为非生成数列，在建模过程中用来获得增量信息。

基本公式：

$$\begin{cases}a^{(1)}(x^{(r)}(k))=x^{(r-1)}(k)\\ a^{(2)}(x^{(r)}(k))=x^{(r-2)}(k)\\ a^{(i)}(x^{(r)}(k))=x^{(r-i)}(k)\\ a^{(r)}(x^{(r)}(k))=x^{(0)}(k)\end{cases} \tag{8.3}$$

换算公式：

$$\begin{cases}x^{(0)}(k)=x^{(1)}(k)-x^{(1)}(k-1)\\ x^{(r-1)}(k)=x^{(r)}(k)-x^{(r)}(k-1)\end{cases} \tag{8.4}$$

8.1.2 GM 模型

GM 模型即灰色模型（Grey Model）。一般建模是用数据列建立差分方程；灰色建模则是用原始数据列做生成后建立微分方程。由于系统被噪声污染了，所以原始数列呈现出离乱的情况。离乱的数列即灰色数列，或者灰色过程，对灰色过程建立的模型，便称为灰色模型。

灰色系统理论（也可称为灰色理论）其所以能够建立微分方程的模型，是基于下述概念、观点、方式和方法：

（1）灰色理论将随机量当做是在一定范围内变化的灰色量，将随机过程当做是在一定范围、一定时区内变化的灰色过程。

（2）灰色理论将无规律的原始数据经生成后，使其变为较有规律的生成数列（通过数列间各时刻数据的逐个累加以得到新的数据和数列，累加前的数列称为原始数列，累加后的数列称为生成数列，记为 AGO）再建模。所以灰色建模实际上是生成数列模型。

（3）灰色理论按开集拓扑定义了数列的时间测度，进而定义信息浓度，定义灰导数与灰微分方程。

（4）灰色理论通过灰数的不同生成方式、数据的不同取舍、不同级别的残差 GM 模型来调整、修正、提高精度（残差是模型计算值与实际值之差）。

（5）灰色理论的模型在考虑残差 GM 模型的补充和修正后，变成了差分、微分方程。

（6）灰色理论的模型选择是基于关联度的概念和关联度收敛原理，关联度收敛是一种有限范围近似的收敛。

（7）灰色 GM 模型一般采用三种检验，即残差大小（或平均值、或最近一个数据的残差值）的检验、关联度检验、后验差检验。残差大小检验是按点检验，关联度检验是建立的模型与指定函数之间近似性的检验，后验差检验是残差分布统计特性的检验。

（8）对高阶系统建模，灰色理论是通过 GM(M,N) 模型群解决的。GM 模型群即一阶微分方程组，也可以通过多级、多次残差 GM 模型的补充修正来解决。

（9）GM 模型所得数据必须经过逆生成（逆生成是指累减生成，累减是指前后两个数之差。记为 IAGO）作还原后才能用。

设 $X_1^{(0)}$ 为系统特征数据序列，$X_i^{(0)}\ (i=2,3,\cdots,N)$ 为相关因素序列，$X_i^{(1)}$ 为诸 $X_i^{(0)}\ (i=1,2,\cdots,n)$ 的 1-AGO 序列，则称

$$X_1^{(1)} = b_2X_2^{(1)} + b_3X_3^{(1)} + \cdots + b_NX_N^{(1)} + a \tag{8.5}$$

为 GM$(0,N)$ 模型。

GM$(0,N)$ 模型不含导数，因此为静态模型，它形如多元线性回归模型，但与一般的多元线性回归模型有着本质的区别，一般的多元线性回归建模以原始数据序列为基础，GM$(0,N)$ 的建模基础则是原始数据的 1-AGO 序列。

8.2 干扰信号的灰色估计

8.2.1 灰色估计的理论基础

灰色系统理论是处理不确定量的一种有效途径，它需要信息少，通用性好，计算方便。采用灰色系统的方法，对于不确定部分建立灰色模型，从而可以估计出不确定参数。

设系统不确定部分符合匹配条件，即为$bD(x,t)$，其中$D(x,t)$包括两部分：一部分与状态x成比例，一部分与状态无关，具体可描述为：

$$D(x,t)=V_1x_1+V_2x_2+\cdots+V_nx_n+f(t)=Vx+f(t) \tag{8.6}$$

式中

$$V=(V_1\ V_2\cdots V_n) \tag{8.7}$$

$$x^{\mathrm{T}}=(x_1\ x_2\cdots x_n) \tag{8.8}$$

设$V_i(i=1,2,\cdots,n)$及$f(t)$均为慢时间变量，可视V及$f(t)$为常数。显然，如果能辨识出V_i及$f(t)$，则可得出$D(x,t)$与$x_1,x_2,\cdots,x_n$的关系，从而估计出对应各状态x的不确定量$D(x,t)$。灰色系统的研究方法之一就是将原始数据进行处理，称为数的“生成”，由于累加生成能弱化随机性，增强规律性，因而它在灰色系统建模中，具有特殊的地位。

令$x^{(0)}$为原始的离散时间函数：

$$x^{(0)}=(x^{(0)}(1),x^{(0)}(2),\cdots,x^{(0)}(n)) \tag{8.9}$$

若

$$x^{(1)}(k)\overset{\Delta}{=}\sum_{m=1}^{k}x^{(0)}(m) \tag{8.10}$$

则称$x^{(1)}(k)$为$x^{(0)}(k)$的累加生成，记为：

$$\mathrm{AGO}x^{(0)}=x^{(1)} \tag{8.11}$$

按灰色系统理论，采用累加生成方法，可建立类似于GM$(0,N)$模型的$D(x,t)$灰色模型。

令离散时间函数为：

$$D^{(0)}=(D(1)\ D(2)\cdots D(N)) \tag{8.12}$$

$$f^{(0)}=(f(1)\ f(2)\cdots f(N)) \tag{8.13}$$

$$\left.\begin{aligned}x_1^{(0)}&=(x_1(1)\ x_1(2)\cdots x_1(N))\\x_2^{(0)}&=(x_2(1)\ x_2(2)\cdots x_2(N))\\&\ \ \vdots\\x_n^{(0)}&=(x_n(1)\ x_n(2)\cdots x_n(N))\end{aligned}\right\} \tag{8.14}$$

式中，$N\geqslant n+1$。

设$D^{(1)}$，$f^{(1)}$，$x_i^{(1)}(i=1,2,\cdots,n)$为$D^{(0)}$，$f^{(0)}$，$x_i^{(0)}(i=1,2,\cdots,n)$的累加生成数列。

将下述关系：

$$D^{(1)}(x,t)=V_1x_1^{(1)}+V_2x_2^{(1)}+\cdots+V_nx_n^{(1)}+f^{(1)} \tag{8.15}$$

称为不确定部分$D(x,t)$的灰色模型。

对于慢时变干扰部分，可认为：

$$\begin{aligned}f^{(1)}(1)&=f(1)\overset{\Delta}{=}f\\f^{(1)}(2)&=2f(1)\overset{\Delta}{=}2f\\&\ \ \vdots\\f^{(1)}(N)&=(N)f\end{aligned} \tag{8.16}$$

记参数列为：

$$\hat{V}^{\mathrm{T}}=(\hat{V}_1\hat{V}_2\cdots\hat{V}_n\ \hat{f})^{\mathrm{T}} \tag{8.17}$$

记数据矩阵为：

$$B=\begin{bmatrix} x_1^{(1)}(2) & \cdots & x_n^{(1)}(2) & 1 \\ x_1^{(1)}(3) & \cdots & x_n^{(1)}(3) & 2 \\ \vdots & & \vdots & \vdots \\ x_1^{(1)}(N) & \cdots & x_n^{(1)}(N) & N-1 \end{bmatrix} \tag{8.18}$$

采用最小二乘法，若(B^TB)可逆，则有：

$$\hat{V}^T=(B^TB)^{-1}B^TD_N^{(1)} \tag{8.19}$$

其中，

$$D_N^{(1)}=(D^{(1)}(1)D^{(1)}(2)\cdots D^{(1)}(N))^T \tag{8.20}$$

将累加值还原，可得式（8.6）的估计模型：

$$\hat{D}(k)=\hat{V}_1x_1(k)+\hat{V}_2x_2(k)+\cdots+\hat{V}_nx_n(k)+\hat{f} \tag{8.21}$$

考虑由下列N个非线性不确定子系统组成的复合非线性不确定系统：

$$\dot{x}=Ax(t)+bu(t)+bD(x,t) \tag{8.22}$$

式中，$x\in R^n,u\in R$，A为$n\times n$维矩阵，b为n维矩阵，$D(x,t)\in R$

$bD(x,t)$代表系统满足匹配条件的不确定部分，它包括参数不确定与外干扰等。

$$D(x,t)=V_1x_1+V_2x_2+\cdots+V_nx_n+f(t) \tag{8.23}$$

不确定部分的$D(x)$无法直接测量，可由测量数据间接计算估出。

离散化为：

$$D(x,k)=\frac{1}{b}(x(k)-Ax(k)-bu(k)) \tag{8.24}$$

式中，$t=kT$，T为采样周期。

灰色估计器的具体算法如下：

第一步：建立$x_i^{(0)}(k)$原始离散数列：

$$\begin{aligned} &i=1,2,\cdots,n \\ &k=1,2,\cdots,N \\ &N\geq n \end{aligned} \tag{8.25}$$

第二步：计算$x_i^{(1)}(k)$累加离散数列：

$$\begin{aligned} &i=1,2,\cdots,n \\ &k=1,2,\cdots,N \end{aligned} \tag{8.26}$$

第三步：计算

$$B=\begin{bmatrix} x_1^{(1)}(2) & \cdots & x_n^{(1)}(2) & 1 \\ x_1^{(1)}(3) & \cdots & x_n^{(1)}(3) & 2 \\ \vdots & & \vdots & \vdots \\ x_1^{(1)}(N) & \cdots & x_n^{(1)}(N) & N-1 \end{bmatrix} \tag{8.27}$$

式中，B^TB必须可逆，若不可逆，则应适当增加N，直到B^TB可逆。

第四步：根据状态$x^{(0)}(k)$及式（8.25），计算$D^{(0)}(k)$离散数列，其中

$$k=1,2,\cdots,N \tag{8.28}$$

第五步：计算$D^{(1)}(k)$累加离散数列：

$$D_N = (D^{(1)}(1) \quad D^{(1)}(2) \quad \cdots \quad D^{(1)}(N))^{\mathrm{T}} \tag{8.29}$$

第六步：计算不确定参数估计值：

$$\hat{V} = (\boldsymbol{B}_b{}^{\mathrm{T}}\boldsymbol{B}_b)^{-1}\boldsymbol{B}_b{}^{\mathrm{T}}\boldsymbol{D}_N^{(1)} \tag{8.30}$$

$$\hat{V} = (\hat{V}_1 \quad \hat{V}_2 \quad \cdots \quad \hat{V}_n \quad \hat{f}_0)^{\mathrm{T}} \tag{8.31}$$

8.2.2 仿真实例

仿真对象为：

$$G(s) = \frac{133}{s(s+25)}$$

将该传递函数转化为状态方程的形式：

$$\dot{x} = \boldsymbol{A}x + \boldsymbol{b}u + \boldsymbol{b}D(x,t)$$

式中，$\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ 0 & -25 \end{bmatrix}$，$\boldsymbol{b} = \begin{bmatrix} 0 \\ 133 \end{bmatrix}$

由于对象为二阶传递函数，迭代次数可达 $N \geq 2$，取 $N=3$。

输入信号取正弦信号 $u=0.50\sin(t)$，外加干扰取 $D(x,t) = V_1x_1 + V_2x_2 + f$。

取如下4组干扰参数：

$$\boldsymbol{V}_1 = [0.10, 0.10, 0.10]$$
$$\boldsymbol{V}_2 = [0.30, 0.30, 0.50]$$
$$\boldsymbol{V}_3 = [0.50, 0.50, 1.50]$$
$$\boldsymbol{V}_4 = [1.50, 1.50, 5.0]$$

经过三个采样时间，仿真得到干扰参数估计结果：

$$\tilde{V}_1 = [0.1021 \quad 0.0997 \quad 0.0998]$$
$$\tilde{V}_2 = [0.2987 \quad 0.2990 \quad 0.4955]$$
$$\tilde{V}_3 = [0.4919 \quad 0.4933 \quad 1.4686]$$
$$\tilde{V}_4 = [1.3873 \quad 1.3912 \quad 4.5875]$$

由此可见，采用灰色估计的方法可有效地估计出干扰参数。

仿真程序的主程序：chap8_1.m。

```
%Grey model prediction
clear all;close all;
global b

para=[];

BB=zeros(1,3);

ts=0.[illegible]1;
N=3;     %Needing N>=2 is [illegible] number
Time=[ts:ts:ts*N];
```

```
%U..ng grey model to pred. t d1st.rbance
V zeros 1,..;
p.ra ..'
[t,x] ode45 'chap. leq',Timeeject,[^ 0],[,,para V ,
x22 x ,2 '    %Speed va..e

  x1 1,:  x 2   ;  %I   s the first value  not incl.d.ng in.t.al val.e)
   BB [x. 1,: ,1];
   for k 2. :N
     x. [x.;
        x. k 1 :. x k+1 : ],
     BB [BB.
       [x1 k,:  k,.'
   end

   D  .ros N-1 1 ,
  for k 2:1 N+1
    ddx(k   x22 k) x22 k-.   .s;
     . k  ?.50*sin k*t  ;
    D k  1 k* ddx k +I*x22(k    . k ;
   end

  D1 zeros N,1 ;
   D1(1  )(1)+D(.);
   for k 2:.:N
      D. k  D. k 1 +.(k+1 ,
  end

  V  nv BB *BB *BB'*D .
  \ V
```

M 函数程序：chap8_1eq.m

```
function dx DynamicModel t,x,flag,para,\.
global  b
dx zeros(.,' ;

. 25.b ...

M 4;
if M  1
  . .  1 ^..];t ^.1;
```

```
elseif M==2
    [illegible]
elseif M==[illegible]
    [illegible]
elseif M==1
    [illegible]
end

[illegible]   %Create true [illegible]

[illegible]*sin(t)   %Give a [illegible] signal [illegible]
dx(1)=x(2);
dx(2)=[illegible]*x(2)+b*[illegible]+DD;
```

8.3 灰色 PID 控制

8.3.1 灰色 PID 控制的理论基础

灰色系统理论是处理不确定量的一种有效途径，它需要信息少，通用性好，计算方便。采用灰色系统的方法，对于不确定部分建立灰色模型，利用它来使控制系统中的灰量得到一定程度的白化，以提高控制质量及其鲁棒性。

设系统不确定部分符合匹配条件，即为$bD(x,t)$，其中$D(x,t)$包括两部分：一部分与状态x成比例，一部分与状态无关，具体可描述为：

$$D(x,t)=V_1x_1+V_2x_2+\cdots+V_nx_n+f(t)=Vx+f(t) \tag{8.32}$$

其中，

$$V=(V_1\ \ V_2\cdots V_n) \tag{8.33}$$

$$x^{\mathrm{T}}=(x_1\ \ x_2\ \cdots\ x_n) \tag{8.34}$$

设$V_i(i=1,2,\cdots,n)$及$f(t)$均为慢时间变量，可视V_i及$f(t)$为常数。显然，如果能辨识出V_i及$f(t)$，则可得出$D(x,t)$与$x_1,x_2,\cdots,x_n$的关系，从而估计出对应各状态x的不确定量$D(x,t)$。灰色系统的研究方法之一就是将原始数据进行处理，称为数的"生成"。由于累加生成能弱化随机性，增强规律性，因而它在灰色系统建模中，具有特殊的地位。

令$x^{(0)}$为原始的离散时间函数

$$x^{(0)}=(x^{(0)}(1),x^{(0)}(2),\cdots,x^{(0)}(n)) \tag{8.35}$$

若

$$x^{(1)}(k)\overset{\Delta}{=}\sum_{m=1}^{k}x^{(0)}(m) \tag{8.36}$$

则称$x^{(1)}(k)$为$x^{(0)}(k)$的累加生成，记为：

$$\mathrm{AGO}x^{(0)}=x^{(1)} \tag{8.37}$$

按灰色系统理论，采用累加生成方法，可建立类似于GM(0,N)模型的$D(x,t)$灰色模型。

令离散时间函数为：

$$D^{(0)}=(D(1)\ \ D(2)\ \cdots D(N)) \tag{8.38}$$

$$f^{(0)}=(f(1)\ \ f(2)\ \cdots f(N)) \tag{8.39}$$

$$\left.\begin{aligned} x_1^{(0)}&=(x_1(1)\ \ x_1(2)\cdots x_1(N))\\ x_2^{(0)}&=(x_2(1)\ \ x_2(2)\cdots x_2(N))\\ &\ \ \vdots\\ x_n^{(0)}&=(x_n(1)\ \ x_n(2)\cdots x_n(N))\end{aligned}\right\} \tag{8.40}$$

式中，$N\geq n+1$

设$D^{(1)}$，$f^{(1)}$，$x_i^{(1)}(i=1,2,\cdots,n)$为$D^{(0)}$，$f^{(0)}$，$x_i^{(0)}(i=1,2,\cdots,n)$的累加生成数列。

将下述关系：

$$D^{(1)}(x,t)=V_1x_1^{(1)}+V_2x_2^{(1)}+\cdots+V_nx_n^{(1)}+f^{(1)} \tag{8.41}$$

称为不确定部分$D(x,t)$的灰色模型

对于慢时变干扰部分，可认为：

$$\begin{aligned} f^{(1)}(1)&=f(1)\overset{\Delta}{=}f\\ f^{(1)}(2)&=2f(1)\overset{\Delta}{=}2f\\ &\ \ \vdots\\ f^{(1)}(N)&=(N)f\end{aligned} \tag{8.42}$$

记参数列为：

$$\hat{V}^{\mathrm{T}}=(\hat{V}_1\ \hat{V}_2\cdots\hat{V}_n\ \hat{f})^{\mathrm{T}} \tag{8.43}$$

记数据矩阵为：

$$\boldsymbol{B}=\begin{bmatrix} x_1^{(1)}(2) & \cdots & x_n^{(1)}(2) & 1\\ x_1^{(1)}(3) & & x_n^{(1)}(3) & 2\\ \vdots & & \vdots & \vdots\\ x_1^{(1)}(N) & \cdots & x_n^{(1)}(N) & N-1\end{bmatrix} \tag{8.44}$$

采用最小二乘法，若$(\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B})$可逆，则有：

$$\hat{V}^{\mathrm{T}}=(\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B})^{-1}\boldsymbol{B}^{\mathrm{T}}D_{TN}^{(1)} \tag{8.45}$$

其中，

$$\boldsymbol{D}_N^{(1)}=(D^{(1)}(1)D^{(1)}(2)\cdots D^{(1)}(N))^{\mathrm{T}} \tag{8.46}$$

将累加值还原，可得式（8.6）的估计模型：

$$\hat{D}(k)=\hat{V}_1x_1(k)+\hat{V}_2x_2(k)+\cdots+\hat{V}_nx_n(k)+\hat{f} \tag{8.47}$$

8.3.2 连续系统灰色PID控制

考虑由下列N个非线性不确定子系统组成的复合非线性不确定系统：

$$\dot{x}=\boldsymbol{A}x(t)+\boldsymbol{b}u(t)+\boldsymbol{b}D(x,t) \tag{8.48}$$

式中，$x\in R^n,u\in R$，$\boldsymbol{A}$为$n\times n$维矩阵，$\boldsymbol{b}$为n维矩阵，$D(x,t)\in R$。

$\boldsymbol{b}D(x,t)$代表系统满足匹配条件的不确定部分，它包括参数不确定与外干扰等。

$$D(x,t) = V_1 x_1 + V_2 x_2 + \cdots + V_n x_n + f(t) \tag{8.49}$$

采用 PID 控制：

$$u(t) = k_p e(k) + k_i \sum_{k=1}^{n} e(k)T + k_d \mathrm{d}e(k) \tag{8.50}$$

为了减弱不确定部分的影响，改善控制性能并提高鲁棒性，在控制器启动过程中，首先采用灰色估计器将不确定部分模型参数 V 粗略地估计出来，然后对 $D(x,t)$ 加以一定程度的补偿。由于这种灰色估计不要求连续实时地进行，故不存在通常实时辨识中存在的数据发散问题。

不确定部分的 $D(x)$ 无法直接测量，可由测量数据间接计算估出。

离散化为：

$$D(x,k) = \frac{1}{b}(\dot{x}(t) - Ax(t) - bu_p(t)) \tag{8.51}$$

式中，$t = kT$，T 为采样周期。

灰色估计器的具体算法如下。

第一步：建立 $x_i^{(0)}(k)$ 原始离散数列。

$$\begin{aligned} &i = 1,2,\cdots,n \\ &k = 1,2,\cdots,N \\ &N \geqslant n \end{aligned} \tag{8.52}$$

第二步：计算 $x_i^{(1)}(k)$ 累加离散数列。

$$\begin{aligned} &i = 1,2,\cdots,n \\ &k = 1,2,\cdots,N \end{aligned} \tag{8.53}$$

第三步：计算

$$\boldsymbol{B} = \begin{bmatrix} x_1^{(1)}(2) & \cdots & x_n^{(1)}(2) & 1 \\ x_1^{(1)}(3) & \cdots & x_n^{(1)}(3) & 2 \\ \vdots & & \vdots & \vdots \\ x_1^{(1)}(N) & \cdots & x_n^{(1)}(N) & N-1 \end{bmatrix} \tag{8.54}$$

式中，$\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B}$ 必须可逆，若不可逆，则应适当增加 N，直到 $\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B}$ 可逆。

第四步：根据状态 $x^{(0)}(k)$ 及式（8.51），计算 $D^{(0)}(k)$ 离散数列，其中

$$k = 1,2,\cdots,N \tag{8.55}$$

第五步：计算 $D^{(1)}(k)$ 累加离散数列：

$$\boldsymbol{D}_N^{(1)} = (D^{(1)}(1) \quad D^{(1)}(2) \quad \cdots \quad D^{(1)}(N)) \tag{8.56}$$

第六步：计算不确定参数估计值：

$$\hat{\boldsymbol{V}} = (\boldsymbol{B}_b^{\mathrm{T}}\boldsymbol{B}_b)^{-1}\boldsymbol{B}_b^{\mathrm{T}}\boldsymbol{D}_N^{(1)} \tag{8.57}$$

$$\hat{V} = (\hat{V}_1 \quad \hat{V}_2 \quad \cdots \quad \hat{V}_n \quad \hat{f}_D)^{\mathrm{T}} \tag{8.58}$$

具有灰色估计器的 PID 控制方法分为以下两个阶段。

第一阶段：采用 PID 进行控制，对不确定部分的模型参数 V 进行估计。

第二阶段：经过 N 步后，在上述控制律基础上，按估计参数 $\hat{\boldsymbol{V}}$ 加上补偿控制 u_c，此时

$$u = u_p + u_c \tag{8.59}$$

$$u_c = -\left[\sum_{i}^{n}\hat{V}_i x_i + \hat{f}\right] \tag{8.60}$$

设

$$\tilde{D}(x,t) \triangleq \sum_{i=1}^{n}(V_i - \hat{V}_i)x_i + (f(t) - \hat{f}) \tag{8.61}$$

采用控制律式（8.59），系统性能将大为改善，鲁棒性大为提高。

8.3.3 仿真程序及分析

仿真实例

设被控制对象为：

$$G(s) = \frac{133}{s(s+25)}$$

将该传递函数转化为状态方程的形式：

$$\dot{x} = Ax + bu + bD(x,t)$$

式中，$A = \begin{bmatrix} 0 & 1 \\ 0 & -25 \end{bmatrix}$，$b = \begin{bmatrix} 0 \\ 133 \end{bmatrix}$。

外加干扰为：

$$D(x,t) = V_1 x_1 + V_2 x_2 + f$$

取干扰参数 $V = [5.0, 5.0, 5.0]$，采用连续系统灰色 PID 预测，经过三个采样时间，得到干扰参数估计结果：

$$\tilde{V} = [4.64215, 5.0129, 5.0597]$$

取 $M=1$，不采用灰色预估补偿，PID 控制跟踪误差及变化率如图 8-1 和图 8-2 所示，取 $M=2$，采用灰色预估补偿，PID 跟踪误差及变化率如图 8-3 和图 8-4 所示。

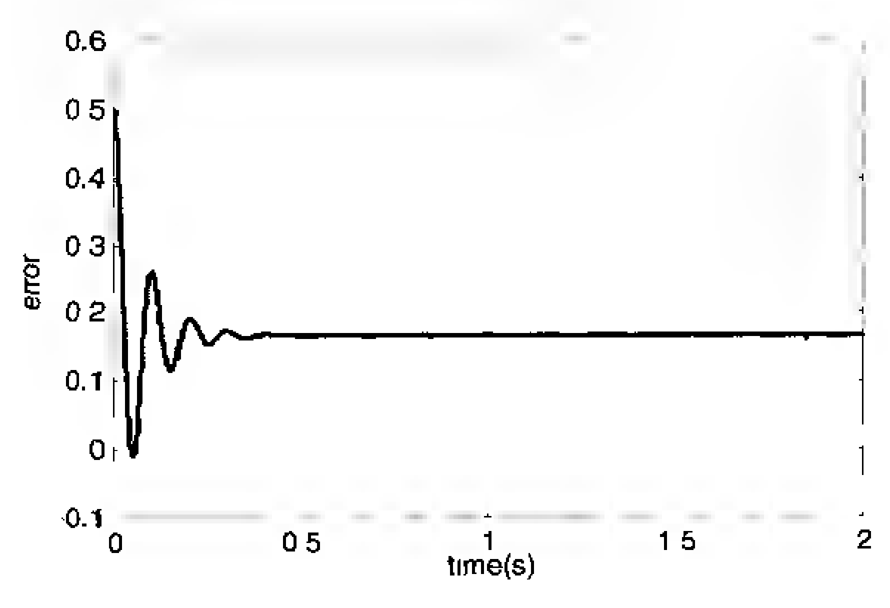


图 8-1 PID 跟踪误差（$M=1$）

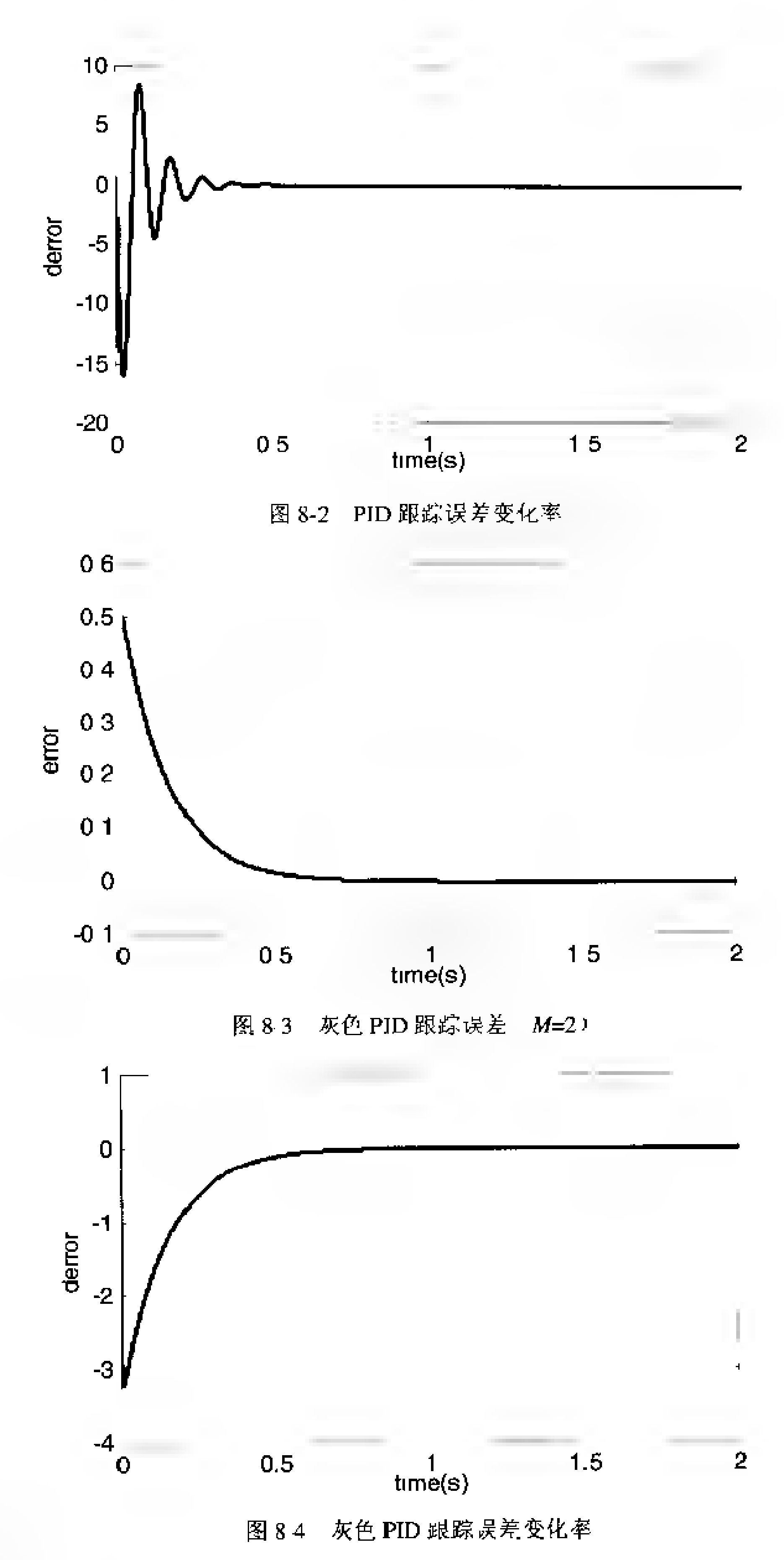


图 8-2　PID 跟踪误差变化率

图 8-3　灰色 PID 跟踪误差（$M=2$）

图 8-4　灰色 PID 跟踪误差变化率

主程序：chap8_2.m。

```
%PID Control based on Grey model compensation
```

```
clear all;close all;
global a21 a22 b A B kp kd

para=[];

BB=zeros(1,3);

ts=0.001;
N=3;    %Needing N>=2(2 is x demension number)
TimeSet=[0:ts:ts*N];

%Using grey model to predict disturbance
V=zeros(1,3);
[t,x]=ode45('chap8_2f',TimeSet,[0.5 0.5],[],para,V);
x11=x(:,1);
x22=x(:,2);

  x1=x(2,:);  %It is the first value(not including initial value)
   BB=[x1(1,:),1];
   for k=2:1:N
        x1=[x1;x1(k-1,:)+x(k+1,:)];
      BB=[BB;[x1(k,:),k]];
  end

   D=zeros(N+1,1);
  for k=2:1:N+1
     ddx(k)=(x22(k)-x22(k-1))/ts;
     up(k)=b*kp*x11(k)+b*kd*x22(k);
     D(k)=1/b*(ddx(k)-a21*x11(k)-a22*x22(k)-up(k));
   end
   D1=zeros(N,1);
   D1(1)=D(1)+D(2);
   for k=2:1:N
      D1(k)=D1(k-1)+D(k+1);
  end

  V1=inv(BB'*BB)*BB'*D1;
   V=V1

%Grey PID control using grey prediction results
N1=2000;
TimeSet1=[0:ts:ts*N1];
[t,x]=ode45('chap8_2f',TimeSet1,[0.50 0.50],[],para,V);
x1=x(:,1);
x2=x(:,2);
```

```
for k=1:1:N1+1
   kpd=[kp kd];
     up(k)=kpd*[x1(k);x2(k)];
     uc(k)=V*[x1(k);x2(k);1];   %Grey Compensation
     u(k)=up(k)+uc(k);
end
figure(1);
plot(t,x1);
xlabel('time(s)');ylabel('error');
figure(2);
plot(t,x2);ylabel('derror');
xlabel('time(s)');ylabel('derror');
figure(3);
plot(t,u);
xlabel('time(s)');ylabel('u');
```

M 函数程序：chap8_2f.m。

```
%Dynamic model
function dx=DynamicModel(t,x,flag,para,V)
global a21 a22 b A B kp kd
dx=zeros(2,1);

a21=0;a22=-25;
b=133;

B=[0;b];
A=[0 1;a21 a22];

V1=[5 5];
f=5;

DD=V1*x+f;     %System true disturbance
%Control law
kp=35;
kd=5;
up=kp*x(1)+kd*x(2);

M=1;
if M==1
   uc=0;           %No Grey Compensation
elseif M==2
   uc=V*[x;1];   %Grey Compensation
end
u=up+uc;
```

```
dx A*x+B* u+DD ;
```

8.3.4 离散系统灰色 PID 控制

直接针对离散系统模型的 PID 控制具有很多对实际应用有价值的优点，但由于不确定部分对控制的影响不能忽略，使控制器难于适应各种不确定性。为此，提出对不确定部分建立灰色估计模型，在有限步数后，根据参数对不确定部分进行一定的补偿，以减小其影响。

考虑单输入离散系统：

$$x(k+1)=Ax(k)+bu(k)+bD(x,k) \tag{8.62}$$

其中，

$$x\in R^n;u\in R;D(x,k)\in R$$

式中，A 为$n\times n$系数矩阵，b 为n维矩阵，$bD(x,k)$代表系统满足匹配条件的不确定部分。它包括参数不确定与外干扰等。

$$D(x,k)=V_1x_1+V_2x_2+\cdots+V_nx_n+d(k) \tag{8.63}$$

控制律分为两个阶段。

(1) 采用 PID 控制进行灰色预测。

$$u(k)=u_{\mathrm{p}}(k) \tag{8.64}$$

$$D(x,k)=\frac{1}{b}(\dot{x}-a_{21}x-a_{22}x-bu_{\mathrm{p}}(t)) \tag{8.65}$$

可计算离散数列向量：

$$D^{(0)}\triangleq(D(0)\quad D(1)\quad\cdots\quad D(N-1))^{\mathrm{T}} \tag{8.66}$$

$$D^{(1)}(k)\triangleq\sum_{l=0}^{k}D(l) \tag{8.67}$$

$$D^{(1)}\triangleq(D^{(1)}(0)\quad D^{(1)}(1)\quad\cdots\quad D^{(1)}(N-1))^{\mathrm{T}} \tag{8.68}$$

由此可见，在 N 步后，即可估计出灰色模型的参数向量$\hat{V}^{\mathrm{T}}$

$$D^{(1)}(x,k)=V_1x_1^{(1)}(k)+V_2x_2^{(1)}(k)+\cdots+V_nx_n^{(1)}(k)+d^{(1)}(k) \tag{8.69}$$

$$\hat{V}=(\hat{V}_1\quad\hat{V}_2\quad\cdots\quad\hat{V}_n\quad\hat{f}_D)^{\mathrm{T}} \tag{8.70}$$

式中

$$d^{(1)}(k)=\sum_{l=0}^{k}d(l) \tag{8.71}$$

$d(k)$为慢时变扰动，在控制过程中可看做不变的常量，将$d(k)$记做d。

按最小二乘法公式，可求得：

$$\hat{V}^{\mathrm{T}}=(B^{\mathrm{T}}B)^{-1}B^{\mathrm{T}}D^{(1)} \tag{8.72}$$

其中，

$$B=\begin{bmatrix}x_1^{(1)}(2) & \cdots & x_n^{(1)}(2) & 1\\ x_1^{(1)}(3) & \cdots & x_n^{(1)}(3) & 2\\ \vdots & & \vdots & \vdots\\ x_1^{(1)}(N) & \cdots & x_n^{(1)}(N) & N-1\end{bmatrix} \tag{8.73}$$

且有：

$$\det(\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B}) > \varepsilon > 0 \tag{8.74}$$

2）采用灰色 PID 控制

在 $(n+1)$ 步后，增加补偿控制 u_c，此时，

$$u = u_p + u_c \tag{8.75}$$

$$u_c = \left[\sum_{i}^{n}\hat{V}_i x_i + \hat{d}\right] \tag{8.76}$$

在第二阶段，估计器停止工作。

8.3.5 仿真程序及分析

仿真实例

被控对象为：

$$G(s) = \frac{133}{s(s+25)}$$

以采样时间为1ms，将传递函数转化为离散状态方程的形式：

$$x(k+1) = Ax(k) + bu(k) + bD(x,k)$$

式中，$A = \begin{bmatrix} 1.0 & 0.0010 \\ 0.0 & 0.9753 \end{bmatrix}$，$b = \begin{bmatrix} 0.0001 \\ -0.1314 \end{bmatrix}$

外加干扰参数 $V = [0.50, 0.50, 0.50]$，采用离散系统灰色 PID 预测，经过 5 个采样时间的 PID 控制，得到干扰估计结果为：$\hat{V} = [0.50000026835824, 0.50000000007557, 0.49999987408683]$。

取 $M=1$，不采用灰色预估补偿，而只采用 PID 控制，跟踪误差及变化率如图 8-5 和图 8-6 所示。取 $M=2$，采用灰色预估补偿，跟踪误差、误差变化率及控制器输出如图 8-7～图 8-9 所示。

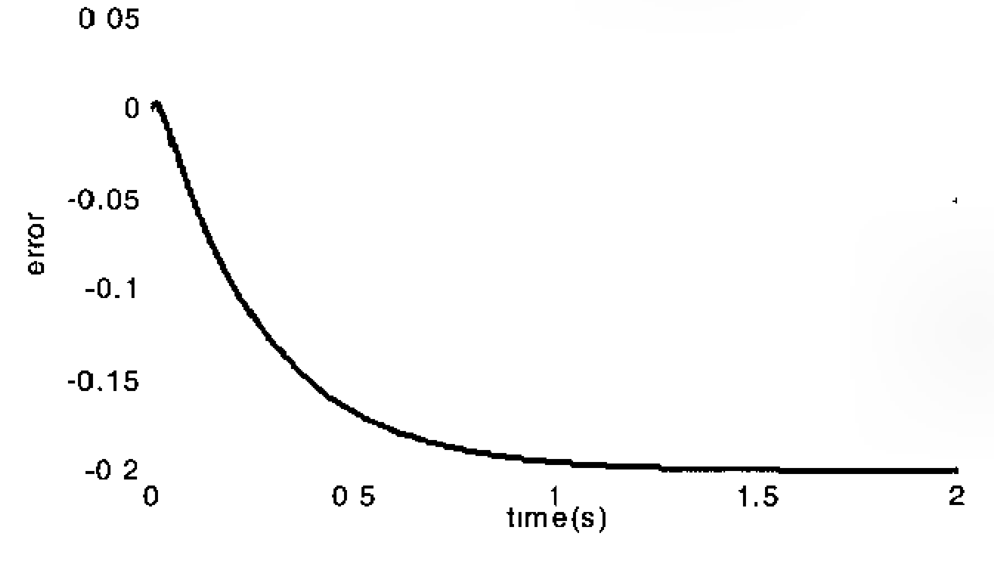


图 8-5　PID 跟踪误差（M=1）

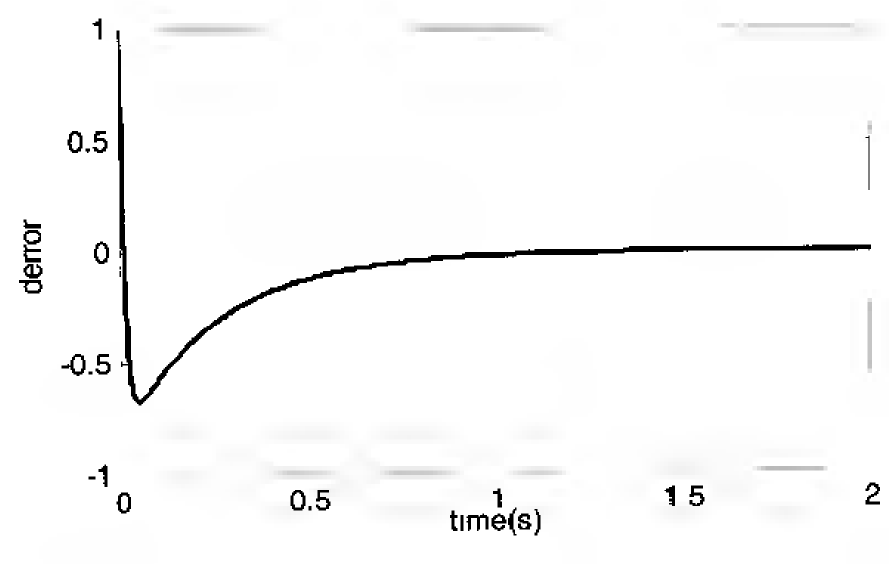


图 8-6　PID 跟踪误差变化率

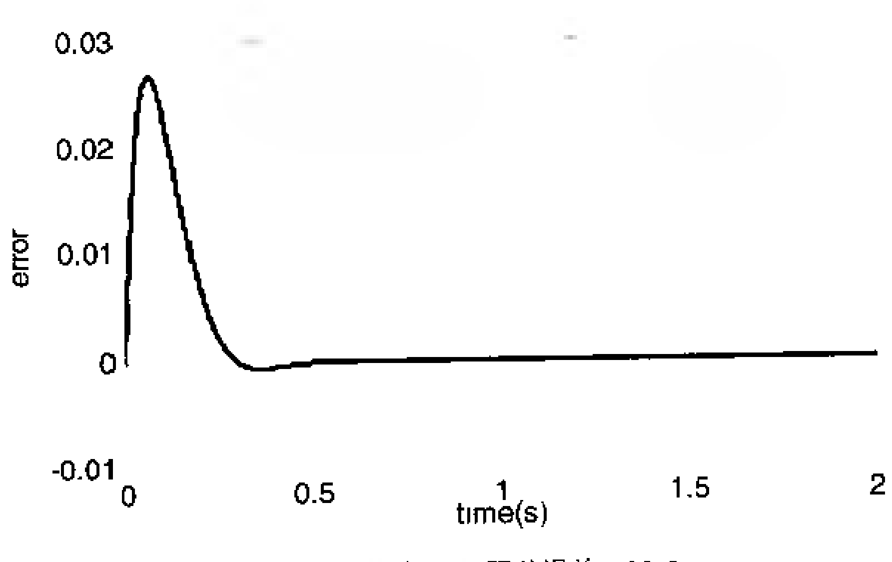


图 8-7　灰色 PID 跟踪误差（$M=2$）

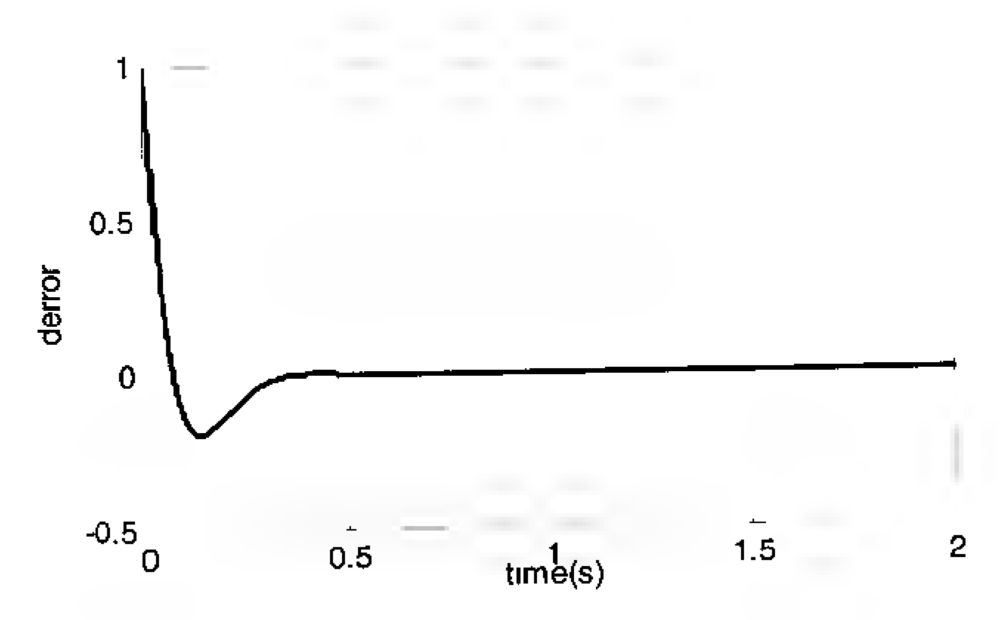


图 8-8　灰色 PID 跟踪误差变化率

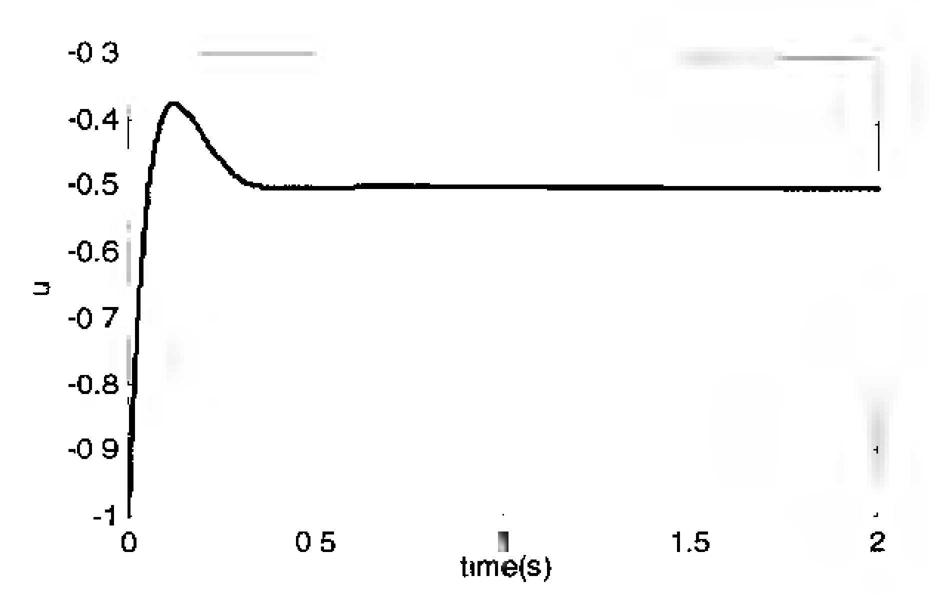


图 8.9 灰色 PID 控制器输出

仿真程序：chap8_3.m。

```
%Discreted PID with grey model prediction
clear all;
close all;
ts=0.001;

n=2;
N=n+1;

a=25;b=133;
J=1/133;
q=25/133;
sys=tf(1,[J,q,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

A1=[0,1;0,-a];
b1=[0;b];
C1=[1,0];
D1=0;
[A,b,C,D]=c2dm(A1,b1,C1,D1,ts,'z');
A=A;
b=b;

x_0=[0;1];
x_1=x_0;

%Uncertain Parameters
```

```
V=[0.5 0.5];d=0.5;
%Initial Value
x_1=[1;1];

%Grey prediction
for k=1:1:N
    time(k)=k*ts;
    x1(k)=x_1(1);
    x2(k)=x_1(2);

    D(k)=V*x_1+d;

    kp=2.0;
    up(k)=-kp*x1(k);
    u(k)=up(k);

    D=V*x_1+d;
    x=A*x_1+b*u(k)+b*D;
    x_1=x;
end
   xx1(1)=x1(2);
   xx2(1)=x2(2);
    BB=[xx1(1) xx2(1) 1];

for i=2:1:N-2
   xx1(i)=xx1(i-1)+x1(i+1);
   xx2(i)=xx2(i-1)+x2(i+1);
   BB=[BB;xx1(i) xx2(i) 1];
end

for i=1:1:N-1
   D(i)=1/b*([x1(i+1);x2(i+1)]-A*[x1(i);x2(i)]-b*up(i));
end
   D1(1)=D(2);
for i=2:1:N-2
   D1(i)=D1(i-1)+D(i+1);
end

%abs(det(BB'*BB));
V1=inv(BB'*BB)*BB'*D1';
Vp=V1

%Grey PID control
```

```
x_1=x_0;
N1=2000;
for k=1:1:N1
time(k)=k*ts;
x1(k)=x_1(1);
x2(k)=x_1(2);

D(k)=V*x_1+d;

%Control law
M=1;
if M==1      %No Grey Compensation
   uc(k)=0;
elseif M==2  %Grey compensation
   uc(k)=-(Vp(1)*x_1(1)+Vp(2)*x_1(2)+Vp(3));
end
up(k)=kp*x1(k);
u(k)=up(k)+uc(k);

D=V*x_1+d;
x=A*x_1+b*u(k)+b*D;
x_1=x;

end
figure(1);
plot(time,x1);
xlabel('time(s)');ylabel('error');
figure(2);
plot(time,x2);
xlabel('time(s)');ylabel('derror');
figure(3);
plot(time,u);
xlabel('time(s)');ylabel('u');
```

8.4 灰色 PID 的位置跟踪

8.4.1 连续系统灰色 PID 位置跟踪

考虑单输入连续系统：

$$\dot{x}=Ax(t)+bu(t)+bD(x,t) \tag{8.77}$$

具中，

$$x\in R^n,\quad u\in R,\quad D(x,t)\in R$$

式中，A 为 $n\times n$ 维矩阵，b 为 n 维矩阵，$bD(x,t)$ 代表系统满足匹配条件的不确定部分，它包括参数不确定与外干扰等。

$$D(x,t)=V_1x_1+V_2x_2+\cdots+V_nx_n+f(t) \tag{8.78}$$

取输入信号为：

$$r(t)=0.5\sin(2\pi Ft) \tag{8.79}$$

离散化：

$$r(k)=0.5\sin(2\pi FkT) \tag{8.80}$$

式中，F 为输入信号频率，T 为采样时间。

令

$$\boldsymbol{R}^{\mathrm{T}}(t)=[r(t)\quad \dot{r}(t)] \tag{8.81}$$

控制律分为以下两个阶段。

（1）采用 PID 控制进行灰色预测

$$u(t)=u_{\mathrm{p}}(t) \tag{8.82}$$

$$\boldsymbol{D}(x,k)=\frac{1}{b}(\ddot{y}+a\dot{y}-bu) \tag{8.83}$$

计算离散数列向量

$$\boldsymbol{D}^{(0)}\overset{\Delta}{=}(D(1)\quad D(2)\quad\cdots\quad D(N))^{\mathrm{T}} \tag{8.84}$$

$$D^{(1)}(k)\overset{\Delta}{=}\sum_{l=0}^{k}D(l) \tag{8.85}$$

$$\boldsymbol{D}^{(1)}\overset{\Delta}{=}(D^{(1)}(2)\quad D^{(1)}(3)\quad\cdots\quad D^{(1)}(N))^{\mathrm{T}} \tag{8.86}$$

在 N 步后，即可估计出灰色模型的参数向量 $\hat{\boldsymbol{V}}^{\mathrm{T}}$。

$$\boldsymbol{D}^{(1)}(x,t)=V_1x_1^{(1)}(t)+V_2x_2^{(1)}(t)+\cdots+V_nx_n^{(1)}(t)+f^{(1)}(t) \tag{8.87}$$

$$\hat{\boldsymbol{V}}=(\hat{V}_1\quad\hat{V}_2\quad\cdots\quad\hat{V}_n\quad\hat{f})^{\mathrm{T}} \tag{8.88}$$

其中，

$$f^{(1)}(k)=\sum_{l=0}^{k}f(l) \tag{8.89}$$

$f(t)$ 为慢时变扰动，在控制过程中可看做不变的常量，将 $f(t)$ 记做 f。

按最小二乘法公式，求得：

$$\hat{\boldsymbol{V}}^{\mathrm{T}}=(\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B})^{-1}\boldsymbol{B}^{\mathrm{T}}\boldsymbol{D}^{(1)} \tag{8.90}$$

其中，

$$\boldsymbol{B}=\begin{bmatrix} x_1^{(1)}(2) & \cdots & x_n^{(1)}(2) & 1 \\ x_1^{(1)}(3) & \cdots & x_n^{(1)}(3) & 2 \\ \vdots & & \vdots & \vdots \\ x_1^{(1)}(N) & \cdots & x_n^{(1)}(N) & N-1 \end{bmatrix} \tag{8.91}$$

且有：

$$\left|\det(\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B})\right|>\varepsilon>0 \tag{8.92}$$

（2）采用灰色 PID 控制。

加入补偿控制 u_c，此时，

$$u = u_p + u_c \tag{8.93}$$

$$u_c = \left[\sum_{i=1}^{n}\hat{V}_i x_i + \hat{f}\right] \tag{8.94}$$

在第二阶段，估计器停止工作。

控制系统状态方程为：

$$x = Ax + bu + bD(x,t) \tag{8.95}$$

$$D(x,t) = V_1 x_1 + V_2 x_2 + f \tag{8.96}$$

采用带有灰色估计器的补偿 PID 控制：

$$u = u_p + u_c \tag{8.97}$$

不加灰色估计器，只用 PID 控制：

$$u = u_p \tag{8.98}$$

8.4.2 仿真程序及分析

仿真实例

考虑单输入单输出连续系统

$$\dot{x} = Ax(t) + bu(t) + bD(x,t)$$

式中，$A = \begin{bmatrix} 0 & 1 \\ 0 & -25 \end{bmatrix}$，$b^{\mathrm{T}} = (0 \quad 133)$

外加干扰参数为 $V = [5, -5]$，$d = 5$，经过 4 个采样时间，干扰参数估计结果为 $V = [4.7117, 5.0109, 5.2018]$。

指令信号为一个幅值为 0.50，频率为 1.0 的正弦信号。取 $M = 1$，不采用灰色预估补偿，即 $u_c(k) = 0$，位置跟踪如图 8-10 所示。取 $M = 2$，采用灰色预估补偿，位置跟踪如图 8-11 所示。

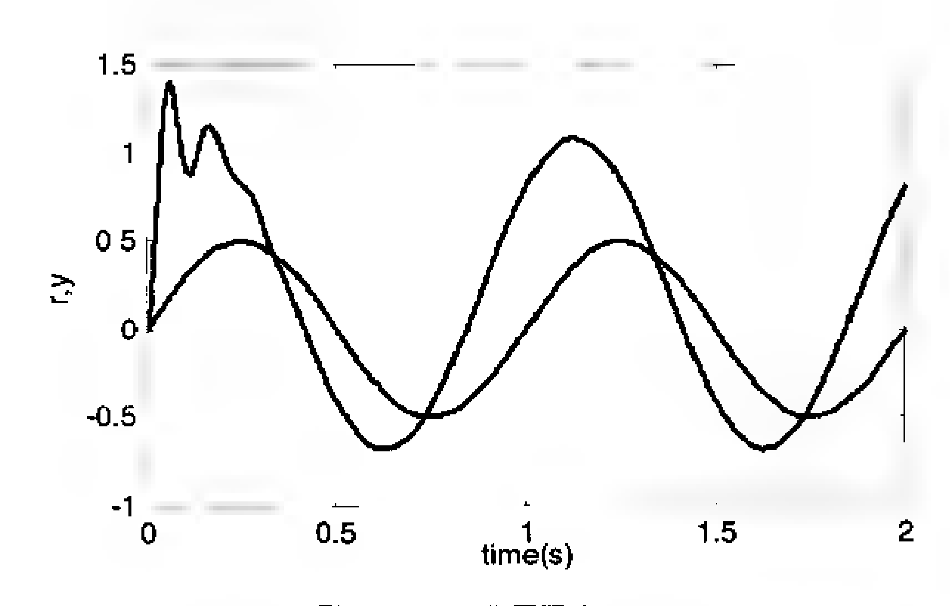


图 8-10 PID 位置跟踪（M=1）

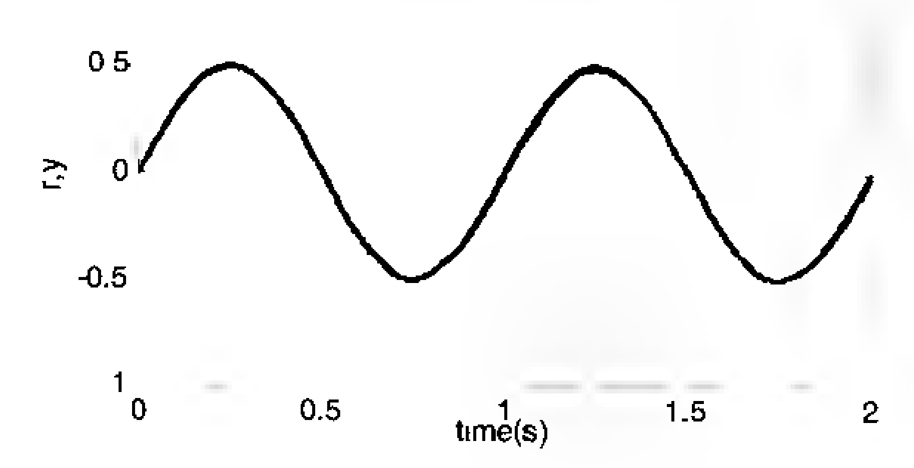



图 8-11　灰色 PID 位置跟踪（M=2）

主程序：chap8_4.m。

```
%Grey model PID Control
clear all;close all;
global kp kd a b A B F AA

F=1.0;
para=[];
AA=0.50;

%Disturbance Prediction
BB=zeros(1,3);
ts=0.001;
N=4;
ab=abs(det(BB'*BB));
  TimeSet=[0:ts:ts*N];
  V=zeros(1,3);
   [t,y]=ode45('chap8_4f',TimeSet,[0 0],[],para,V);  %Grey Prediction

  w=2*pi*F;
  r=AA*(sin(w*t));
  dr=AA*w*cos(w*t);

  y0=y;
  y1=y0(2,:);
  BB=[y1(1,:),1];

   for k=2:N
    y1=[y1;y1(k-1,:)+y0(k+1,:)];
    BB=[BB;[y1(k,:) k]];
  end
```

```
y11 y(:,1);
y22 y(:,2 ;

   D-zeros N+1,1),
   for k.2:1:N+1
      D k)-1 b*( y22(k  ,22,k 1   ts a*y22(k  b*kp*(r(k) y11(k))
b*kd*(dr(k  y22 k)) ,
    end

   D1 zeros N,1 ;
   D1(1) D(2);

   for k 2:1:N
      I 1,k -D1(k 1 +. k+1);
   end
   ab abs det(BB'*BB))

V1 inv(BB'*BB *BB'*D1;
V V1

%PID Control
N1 2000;
TimeSet1-[0:ts:ts*N1];
[t,y] ode45('chap8_4f',TimeSet1,.0 0],[],para,V),

y1 y :,1);
y2 y(:,2 ;
r AA*(sin(w*t));
dr AA*w*cos(w*t):
ddr -AA*w^2*sin w*t,;

e r y1;
de dr y2;
for k 1:1:N1+1
     up(k) kp*e(k +kd*de(k);
     uc(k)  V*[y1 k);y2 k);1];   %Grey Compensation
     u k) up(k +uc(k ;
end
figure(1 ;grid on;
plot t,r, r',t,y :,1 , b' ;
xlabel( time(s   ;ylabel, r,y' ;

figure 2 ,grid on;
plot t,r y :,. .,
```

```
xlabel('time(s)');ylabel('error');

figure(3);grid on;
plot(t,u);
xlabel('time(s)');ylabel('u');
```

M **函数程序：chap8_4f.m。**

```
%Dynamic model
function dy=DynamicModel(t,y,flag,para,V)
global kp kd a b A B F AA
dy=zeros(2,1);

%Input signal
w=2*pi*F;
rr=AA*sin(w*t);
dr=AA*w*cos(w*t);
ddr=-AA*(w^2)*sin(w*t);

r=[rr;dr];

b=133;a=-25;
B=[0;b];
A=[0 1;0 a];

V1=[5 5];f=5;

DD=V1*y+f;      %True disturbance
%Control law
kp=30;kd=5.0;kpd=[kp,kd];
up=kpd*(r-y);

M=2;
if M==1
   uc=0;          %No Grey Compensation
elseif M==2
   uc=-V*[y;1];   %Grey Compensation
end
u=up+uc;
dy=A*y+B*(u+DD);
```

8.4.3 离散系统灰色PID位置跟踪

考虑单输入离散系统：

$$x(k+1)=\boldsymbol{A}x(k)+\boldsymbol{b}u(k)+\boldsymbol{b}D(x,k) \tag{8.99}$$

式中，$x\in R^n, u\in R, D(x,k)\in R$，$\boldsymbol{A}$ 为 $n\times n$ 维矩阵，$\boldsymbol{b}$ 为 n 维矩阵。

$\boldsymbol{b}D(x,k)$ 代表系统满足匹配条件的不确定部分，它包括参数不确定与外干扰等。

$$D(x,k)=V_1x_1+V_2x_2+\cdots+V_nx_n+d(k) \tag{8.100}$$

取输入信号为：

$$r(t)=0.5\sin(2\pi Ft) \tag{8.101}$$

离散化为：

$$r(k)=0.5\sin(2\pi FkT) \tag{8.102}$$

式中，F 为输入信号频率，T 为采样时间。

令

$$\boldsymbol{R}^{\mathrm{T}}(k)=[r(k)\quad \dot{r}(k)] \tag{8.103}$$

控制律分为以下两个阶段：

（1）采用 PID 控制进行灰色预测

$$u(k)=u_{\mathrm{p}}(k) \tag{8.104}$$

$$D(x,k)=b^{+}(x(k+1)-\boldsymbol{A}x(k)-\boldsymbol{B}u(k)) \tag{8.105}$$

可计算离散数列向量：

$$\boldsymbol{D}^{(0)}\overset{\Delta}{=}(D(0)\quad D(1)\quad \cdots\quad D(N-1))^{\mathrm{T}} \tag{8.106}$$

$$D^{(1)}(k)\overset{\Delta}{=}\sum_{l=0}^{k}D(l)$$

$$\boldsymbol{D}^{(1)}\overset{\Delta}{=}(D^{(1)}(0)\quad D^{(1)}(1)\quad \cdots\quad D^{(1)}(N-1))^{\mathrm{T}} \tag{8.107}$$

由此可见，经过 N 步后，即可估计出灰色模型

$$D^{(1)}(x,k)=V_1x_1^{(1)}(k)+V_2x_2^{(1)}(k)+\cdots+V_nx_n^{(1)}(k)+d^{(1)}(k) \tag{8.108}$$

的参数向量 $\hat{\boldsymbol{V}}^{\mathrm{T}}$：

$$\hat{\boldsymbol{V}}=(\hat{V}_1\quad \hat{V}_2\quad \cdots\quad \hat{V}_n\quad \hat{f}_{\mathrm{D}})^{\mathrm{T}} \tag{8.109}$$

其中，

$$d^{(1)}(k)=\sum_{l=0}^{k}d(l) \tag{8.110}$$

$d(k)$ 为慢时变扰动，可看做不变的常量，将 $d(k)$ 记做 d。

按最小二乘法公式求得：

$$\hat{\boldsymbol{V}}^{\mathrm{T}}=(\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B})^{-1}\boldsymbol{B}^{\mathrm{T}}\boldsymbol{D}^{(1)} \tag{8.111}$$

其中，

$$\boldsymbol{B}=\begin{bmatrix} x_1^{(1)}(2) & \cdots & x_n^{(1)}(2) & 1 \\ x_1^{(1)}(3) & \cdots & x_n^{(1)}(3) & 2 \\ \vdots & & \vdots & \vdots \\ x_1^{(1)}(N) & \cdots & x_n^{(1)}(N) & N-1 \end{bmatrix} \tag{8.112}$$

且有：

$$\left|\det(\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B})\right|>\varepsilon>0 \tag{8.113}$$

2）采用灰色 PID 控制。

增加补偿控制 u_c，此时控制律为：

$$u = u_p + u_c \tag{8.114}$$

$$u_c = -\left[\sum_{i=1}^{n} \hat{V}_i x_i + \hat{d}\right] \tag{8.115}$$

在第二阶段，估计器停止工作。

8.4.4 仿真程序及分析

仿真实例

考虑单输入连续系统：

$$\dot{x} = Ax + bu + bD(x,t)$$

$$A = \begin{bmatrix} 0 & 1 \\ 0 & -25 \end{bmatrix}, b^T = (0 \quad 133)$$

以采样时间 1ms 离散化为：

$$x(k+1) = Ax(k) + bu(k) + bD(x,k)$$

$$A = \begin{bmatrix} 1.0000 & 0.0010 \\ 0 & 0.9753 \end{bmatrix}, \quad b^T = (0.0001 \quad 0.1314)$$

外加干扰参数为 $V = [5,-5]$，$d = 5$，经过 5 个采样时间，可得到干扰参数估计结果：

$$V_p = [5.0000 \quad -5.0000 \quad 5.0000]$$

指令信号为一个幅值为 0.50，频率为 3.0 的正弦信号。当 $G=1$ 时为 PID 灰色预测，当 $G=2$ 时为 PID 灰色控制。在灰色 PID 控制时，取 $M=1$ 为不采用灰色预估补偿，即 $u_c(k)=0$，位置跟踪如图 8-12 所示；取 $M=2$ 为采用灰色预估补偿，位置跟踪如图 8-13 所示。

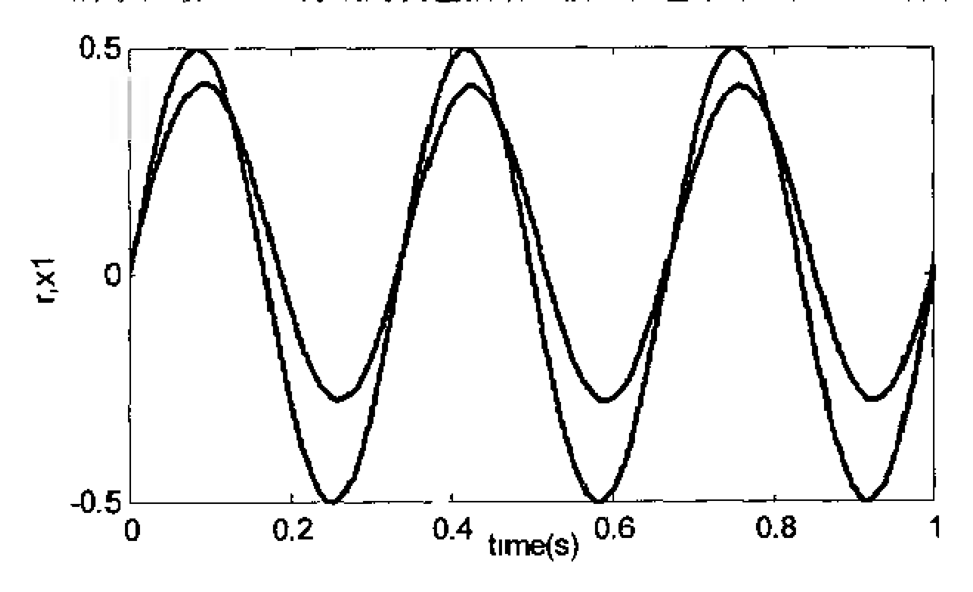


图 8-12　未采用灰色预估补偿 PID 位置跟踪（M=1）

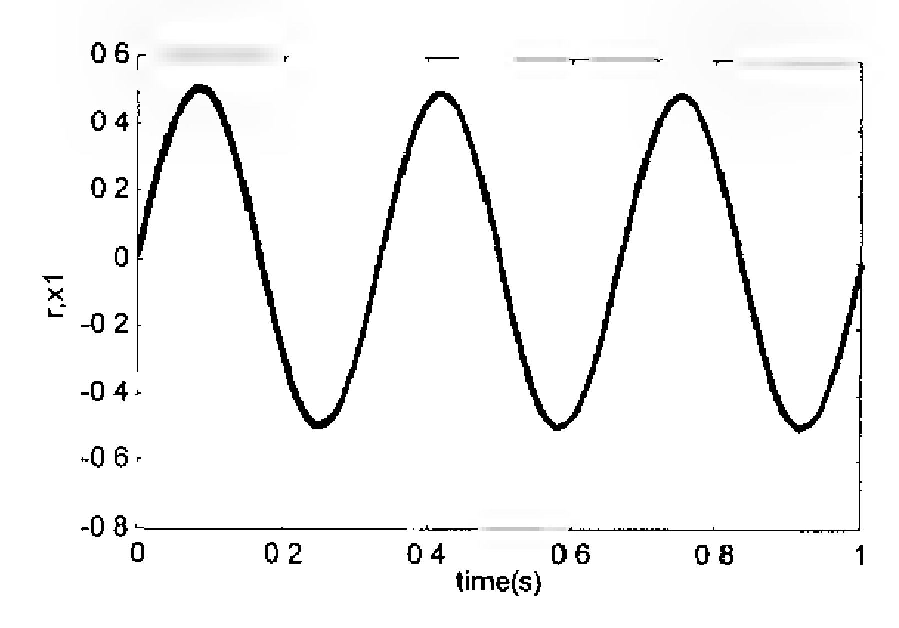


图 8.13　灰色 PID 位置跟踪（$M=2$）

仿真程序：chap8_5.m。

```
%Discrete PID Control with grey model prediction
clear all;
close all;

ts=0.001;
n=2;

AA=0.5;
F=3.0;
N1=1000;

N=n+3;
w=2*pi*F;

%System model
A1=[0 1;0 -25];
B1=[0;133];
C1=[1 0];
D1=[0];
[A,B,C,D]=c2dm(A1,B1,C1,D1,ts,'z');

V=[5 -5],d=5;
x_1=[0;0];
for G=1:2
   for k=1:1:N
   time(k)=k*ts;
```

```
    r(k)=AA*sin(w*k*ts);
    dr(k)=AA*w*cos(w*k*ts);

    x1(k)=x_1(1);
    x2(k)=x_1(2);

%Control law
if G==1          %For Grey Prediction
    uc(k)=0;
elseif G==2      %For Grey PID Control
    M=2;
    if M==1       %No Grey Compensation
        uc(k)=0;
    elseif M==2  %Grey Compensation
        uc(k)=-(Vp(1)*x_1(1)+Vp(2)*x_1(2)+Vp(3));
    end
end

kp=80;kd=10;
e(k)=r(k)-x1(k);
de(k)=dr(k)-x2(k);

up(k)=kp*e(k)+kd*de(k);
u(k)=up(k)+uc(k);

%Plant
    DD=V*x_1+d;
    x=A*x_1+B*u(k)+B*DD;
    x_1=x;
end

if G==1       %Grey prediction
      xx1(1)=x1(2);xx2(1)=x2(2);
      BB=[xx1(1) xx2(1) 1];

for i=2:1:N-2
     xx1(i)=xx1(i-1)+x1(i+1);
     xx2(i)=xx2(i-1)+x2(i+1);
     BB=[BB;xx1(i) xx2(i) 1];
end
```

```
for i=1:N-1
   u(i)=kp*e(i)+kd*de(i);
   DD(i)=1/B*([x1(i+1);x2(i+1)]-A*[x1(i);x2(i)])-u(i);
end
D1(1)=DDD(2);

for i=2:N-1
   D1(i)=D1(i-1)+DDD(i+1);
end
end

xp=[x1  x2];

if G==1
     ab=abs(det(BB'*BB));
     V1=inv(BB'*BB)*BB'*D1'
     Vp=V1'
end

N=N1;  %If G==2
end
figure(1);grid on;
plot(time,r,'r',time,x1,'b');
xlabel('time(s)');ylabel('r,x1');
figure(2);grid on;
plot(time,r-x1,'k');
xlabel('time(s)');ylabel('error');
figure(3);grid on;
plot(time,u);
xlabel('time(s)');ylabel('u');
```

第 9 章　伺服系统 PID 控制

9.1　基于 Lugre 摩擦模型的 PID 控制

9.1.1　伺服系统的摩擦现象

在高精度、超低速伺服系统中，由于非线性摩擦环节的存在，使系统的动态及静态性能受到很大程度的影响，主要表现为低速时出现爬行现象，稳态时有较大的静差或出现极限环振荡。

摩擦现象是一种复杂的、非线性的、具有不确定性的自然现象，摩擦学的研究结果表明，人类目前对于摩擦的物理过程的了解还只停留在定性认识阶段，无法通过数学方法对摩擦过程给出精确描述。在现实生活中，摩擦现象几乎无处不在。在有些情况下，摩擦环节是人们所期望的，如汽车的刹车系统，但对于机械伺服系统而言，摩擦环节却成为提高系统性能的障碍，使系统出现爬行、振荡或稳态误差。为了减轻机械伺服系统中摩擦环节带来的负面影响，人们在大量的实践中总结出很多有效的方法，可概括为三类：

（1）改变机械伺服系统的结构设计，减少传动环节；

（2）选择更好的润滑剂，减小动静摩擦的差值；

（3）采用适当的控制补偿方法，对摩擦力（矩）进行补偿。

有关摩擦建模及动态补偿控制技术方面的研究具有近百年的历史，但由于当时控制理论和摩擦学发展水平的限制，使得这方面的研究一直进展不大，进入 20 世纪 80 年代以后，这一领域的研究渐渐活跃，许多先进的摩擦模型和补偿方法被相继提出，其中许多补偿技术已经在机械伺服系统的控制设计中得到了成功的应用。

在伺服系统辨识中，选择一个合适的摩擦模型是非常重要的，实践表明，采用简单的库仑摩擦+粘性摩擦作为摩擦模型，其效果并不理想。目前，已提出的摩擦模型很多，主要有 Karnopp 模型、Lugre 模型及综合模型。其中，Lugre 模型是 Canudas 等在 1995 年提出的典型伺服系统的摩擦模型[34]，该模型能够准确地描述摩擦过程的复杂的动态、静态特性，如爬行（Stick Slip）、极限环振荡（Hunting）、滑前变形（Presliding Displacement）、摩擦记忆（Friction Memory）、变静摩擦（Rising Static Friction）及静态 Stribeck 曲线。

9.1.2　伺服系统的 Lugre 摩擦模型

Lugre 摩擦模型可描述如下：

对于伺服系统，用下面的微分方程表示：

$$J\ddot{\theta} = u - F \tag{9.1}$$

式中，J 为转动惯量，θ 为转角，u 为控制力矩，F 为摩擦力矩。设状态变量 z 代表接触面鬃毛的平均变形（Bristle Deform），则 F 可由下面的 Lugre 模型来描述：

$$F = \sigma_0 z + \sigma_1 \dot{z} + \alpha\dot{\theta} \tag{9.2}$$

$$\dot{z}=\dot{\theta}-\frac{\sigma_0|\dot{\theta}|}{g(\dot{\theta})}z \tag{9.3}$$

$$g(\dot{\theta})=F_c+(F_s-F_c)e^{-\left(\frac{\dot{\theta}}{V_s}\right)^2}+\alpha\dot{\theta} \tag{9.4}$$

在式（9.2）～式（9.4）中，σ_0、σ 称为动态摩擦参数，F_c、F_s、α、V_s 称为静态摩擦参数，其中 F_c 为库仑摩擦，F_s 为静摩擦，α 为粘性摩擦系数，V_s 为切换速度。

控制器采用 PD 控制的形式。

9.1.3 仿真程序及分析

仿真实例

在伺服系统（式（9.1））及摩擦模型（式（9.2）～式（9.4））中，取 J=1.0，σ_0=260，σ_1=2.5，α=0.02，F_c=0.28，F_s=0.34，V_s=0.01。取输入信号为正弦信号。

仿真方法一：采用 M 语言实现控制算法及带有摩擦模型的被控对象的描述

采用 PD 控制，取 $k_p=50$；$k_d=0.010$。图 9-1 和图 9-2 为位置和速度跟踪的仿真结果。在速度过零点时，波形发生畸变，出现位置跟踪"平顶"现象和速度跟踪"死区"现象。

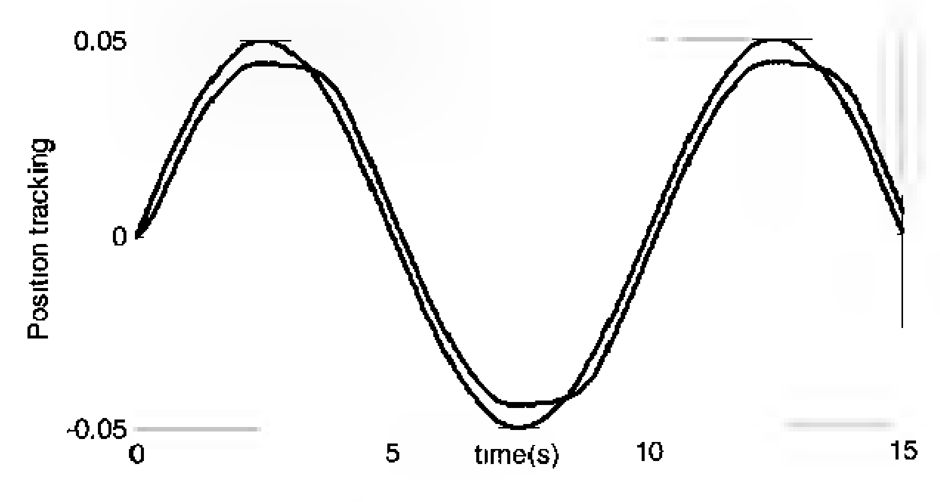


图 9-1　PID 的位置跟踪

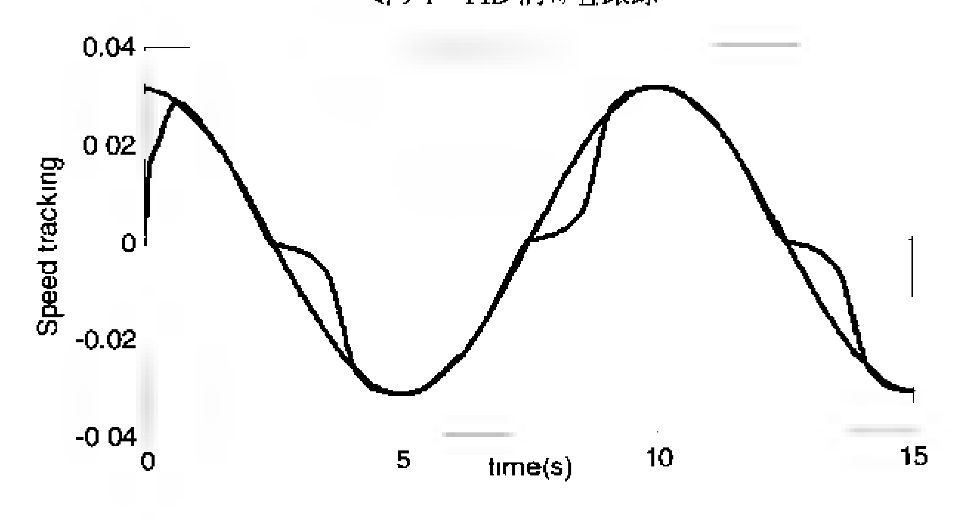


图 9-2　PID 的速度跟踪

主程序：chap9_1.m。

```
%PID Control based on Lugre friction model
clear all;
close all;

ts=[illegible];
u_1=[illegible];
qq=zeros([illegible],1);

for k=1:1:[illegible]

time(k)=k*ts;
rin(k)=0.05*sin(0.1*2*pi*k*ts);
drin(k)=0.05*0.1*2*pi*cos(0.1*2*pi*k*ts);

%Lugre friction model
[tt,yy]=ode15s('chap9_1eq',[0 ts],qq,[],u_1);
qq=yy(length(yy),:)';
yout(k)=qq(1);
y2(k)=qq(2);

u(k)=50*(rin(k)-yout(k))+0.010*(drin(k)-y2(k))/ts;
u_1=u(k);
end
figure(1);
plot(time,rin,'r',time,yout,'b');
xlabel('time(s)');ylabel('Position tracking');
figure(2);
plot(time,drin,'r',time,y2,'b');
xlabel('time(s)');ylabel('Speed tracking');
```

摩擦模型子程序：chap9_1eq.m。

```
function dy=dym(t,y,flag,u)

c0=260;
c1=[illegible];
c2=0.02;
Fc=0.28;
Fs=0.34;
Vs=0.01;
J=1.0;
```

仿真方法二：采用 Simulink 仿真

在 S 函数中实现 Lugre 摩擦模型的描述，该仿真可很容易地给出位置跟踪、摩擦力矩和控制力矩的变化，且运行速度快。位置指令为 $r(t)=0.1\sin(0.2\pi t)$(度)。采用 PD 控制，取 $k_p=20$，$k_d=5.0$。仿真时间为 20s，[illegible]，仿真结果如图 9-3～图 9-6 所示。

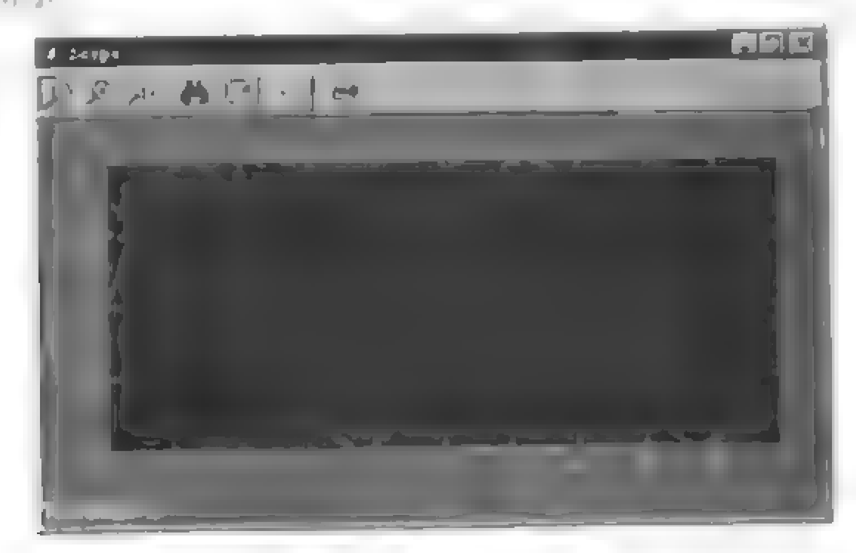


图 9-3 PID 的[illegible]

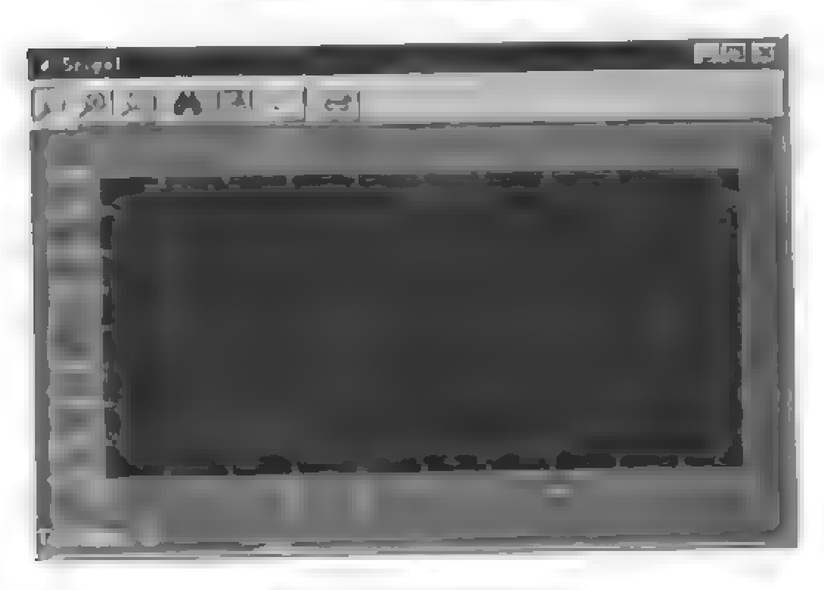


图 9-4 PID [illegible]

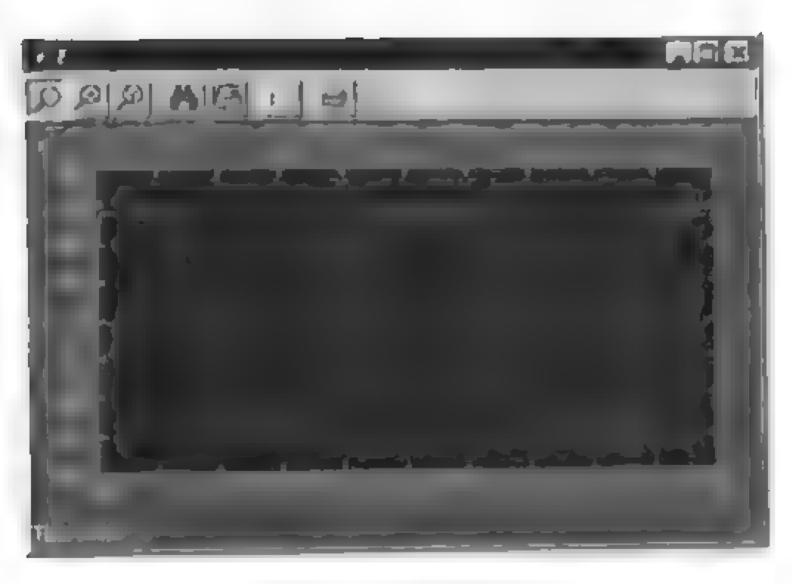


图 9-5　摩擦力矩的变化

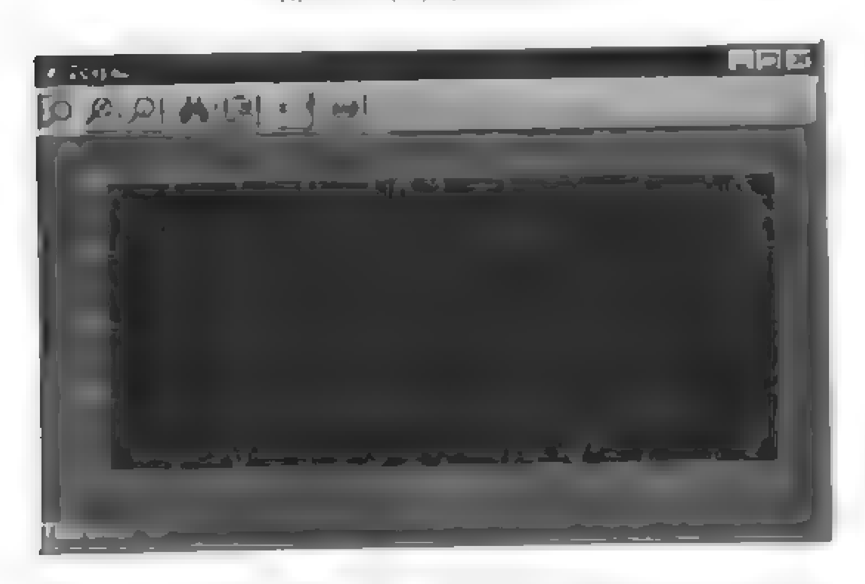


图 9-6　控制力矩的变化

初始化程序：chap9_2i.m

S 函数摩擦模型程序：chap9_2s.m

```
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end

function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 3;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 3;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1; % At least one sample time is needed
sys = simsizes(sizes);
x0  = [0;0;0];
str = [];
ts  = [0 0];

function sys=mdlDerivatives(t,x,u)  %LuGre model
global J rou0 rou1 af

Fc=0.28;
Fs=0.34;
vs=0.01;

%Ref:pid_fm_eq.m
g=Fc+(Fs-Fc)*exp(-(x(2)/vs)^2)+af*x(2);
sys(3)=x(2)-rou0*abs(x(2))/g*x(3);
F=rou0*x(3)+rou1*sys(3)+af*x(2);
sys(1)=x(2);
sys(2)=1/J*(u-F);  %Important!

function sys=mdlOutputs(t,x,u)
sys(1)=x(1);   %Angle
sys(2)=x(2);   %Angle speed
sys(3)=x(3);   %z
```

Simulink 主程序：chap9_2.mdl，如图 9-7 和图 9-8 所示

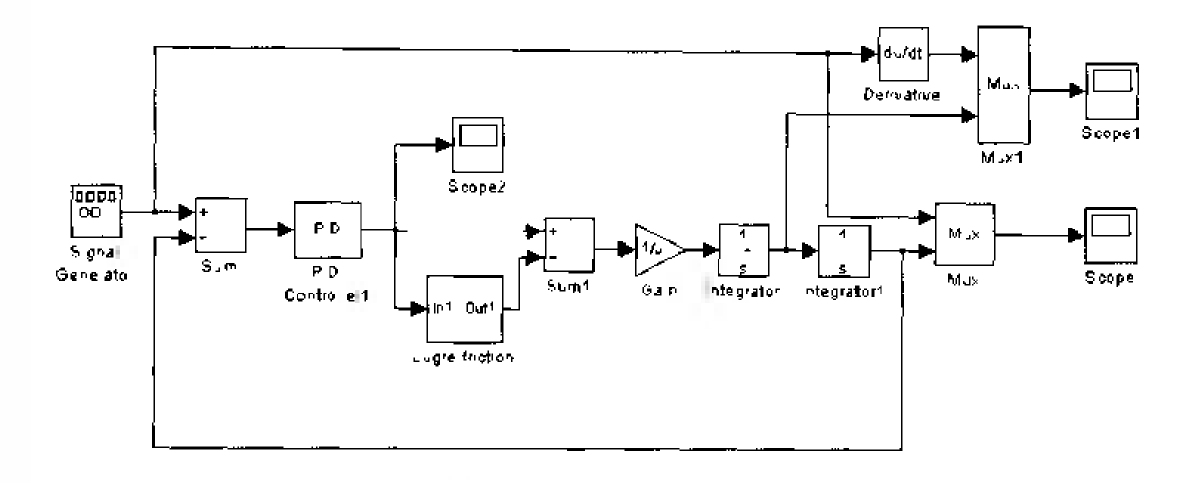


图 9-7 Simulink 主程序

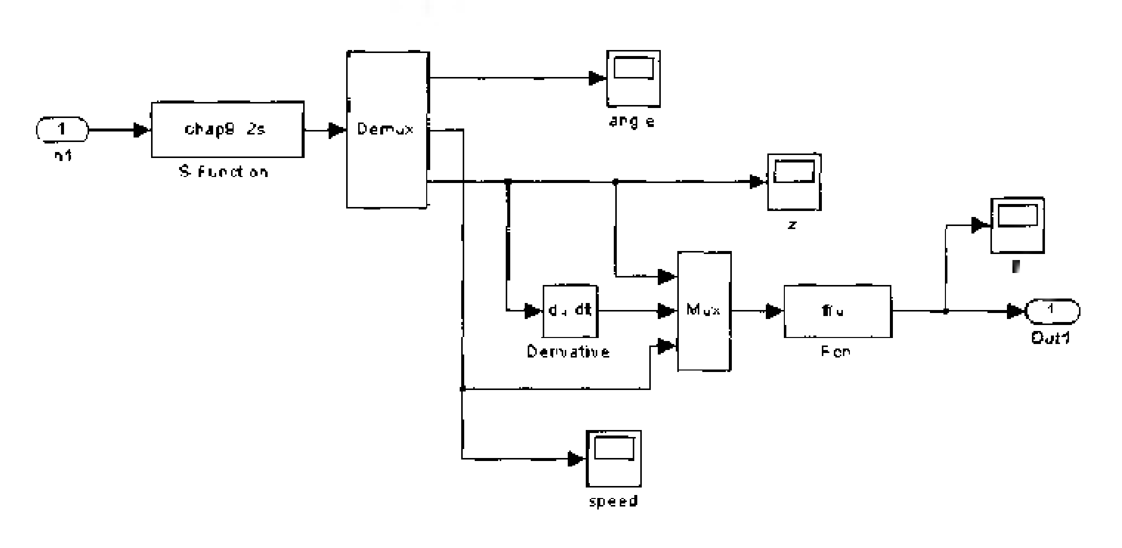


图 9-8 摩擦模型的 Simulink 子程序

9.2 基于 Stribeck 摩擦模型的 PID 控制

9.2.1 Stribeck 摩擦模型描述

Stribeck 曲线是比较著名的摩擦模型[35]。如图 9-9 所示，该图表明在不同的摩擦阶段，摩擦力矩与速度之间的关系，该关系即为 Stribeck 曲线。

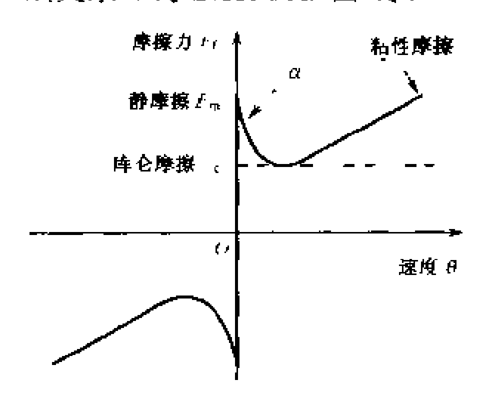


图 9-9 摩擦-速度稳态关系曲线（Stribeck 曲线）

Stribeck 摩擦模型可表示为：

当 $|\dot{\theta}(t)|<\alpha$ 时，静摩擦为：

$$F_f(t)=\begin{cases}F_m & F(t)>F_m\\ F(t) & -F_m<F<F_m\\ -F_m & F(t)<-F_m\end{cases} \tag{9.5}$$

当 $|\dot{\theta}(t)|>\alpha$ 时，动摩擦为：

$$F_f(t)=\left(F_c+(F_m-F_c)e^{-\alpha_1|\dot{\theta}(t)|}\right)\operatorname{sgn}\left(\dot{\theta}(t)\right)+k_v\dot{\theta} \tag{9.6}$$

$$F(t)=J\ddot{\theta}(t) \tag{9.7}$$

式中，$F(t)$ 为驱动力，F_m 为最大静摩擦力，F_c 为库仑摩擦力，k_v 为粘性摩擦力矩比例系数，$\dot{\theta}(t)$ 为转动角速度，α 和 α_1 为非常小的、正的常数

9.2.2 一个典型伺服系统描述

以飞行模拟转台伺服系统为例，它是三轴伺服系统，在正常情况下可简化为线性二阶环节的系统，在低速情况下具有较强的摩擦现象，此时控制对象变为非线性，很难用传统控制方法达到高精度控制。任意框的伺服结构可表达为如图 9.10 所示的形式，该系统采用直流电机，忽略电枢电感，电流环和速度环为开环

其中 K_u 为 PWM 功率放大器放大系数，R 为电枢电阻，K_m 为电机力矩系数，C_e 为电压反馈系数，J 为该框的转动惯量，$\dot{\theta}(t)$ 为转速，$r(t)$ 为指令信号，$u(t)$ 为控制输入。

根据伺服系统的结构，飞行模拟转台位置状态方程可描述如下：

$$\begin{bmatrix}\dot{x}_1(t)\\ \dot{x}_2(t)\end{bmatrix}=\begin{bmatrix}0 & 1\\ 0 & -\dfrac{K_mC_e}{JR}\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix}+\begin{bmatrix}0\\ K_u\dfrac{K_m}{JR}\end{bmatrix}u(t)-\begin{bmatrix}0\\ \dfrac{1}{J}\end{bmatrix}F_f(t) \tag{9.8}$$

式中，$x_1(t)=\theta(t)$ 为转角、$x_2(t)=\dot{\theta}(t)$ 为转速

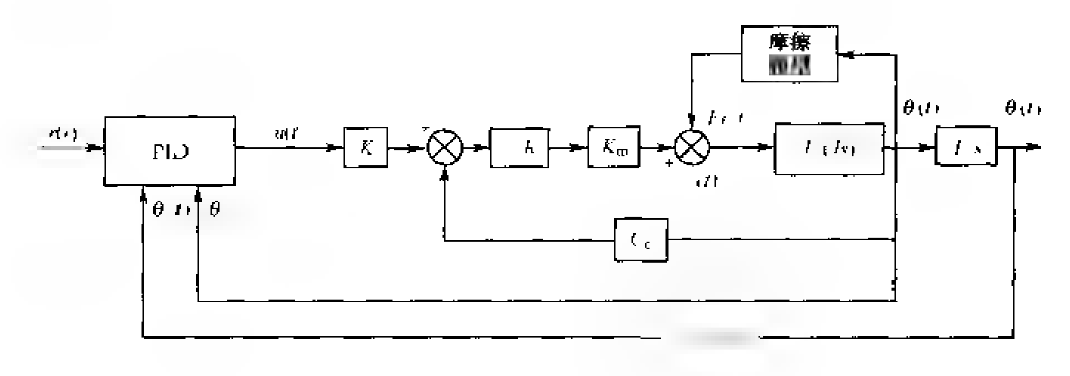


图 9.10 飞行模拟转台伺服系统结构

9.2.3 仿真程序及分析

仿真实例

设某转台某框伺服系统参数如下：

$R = 7.77\Omega$、$K_m = 6\text{N}\cdot\text{m/A}$，$C_e = 1.2\text{V/(rad/s)}$，$J = 0.6\text{kg}\cdot\text{m}^2$，

$K_u = 11\text{V/V}$, $F_c = 15\text{N}\cdot\text{m}$，$F_m = 20\text{N}\cdot\text{m}$, $k_v = 2.0\text{Nms/rad}$, $a_1 = -1.0$, $\alpha = 0.01$

S 为信号选择变量，其中 $S=1$ 为正弦信号，$S=2$ 为阶跃信号，$S=3$ 为方波信号。本仿真选取 $S=1$，低速正弦跟踪信号指令为 $r(t) = 0.10\sin(2\pi t)$。

采用 PD 控制，

$$u(t) = 200e(t) + 40\dot{e}(t)$$

针对仿真方法一，假设伺服系统无摩擦，即取 $M=0$，此时 $F_f(t)=0$，采用 PD 控制，速度和位置跟踪结果如图 9-11 和图 9-12 所示。仿真结果表明，无摩擦时，采用 PID 控制方法具有较好的控制效果。

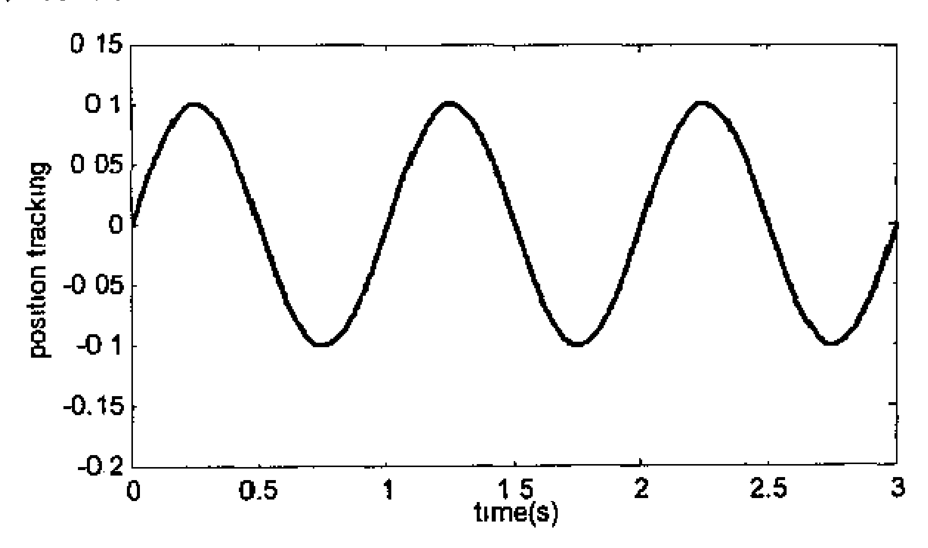


图 9-11 无摩擦时的位置跟踪（M=0）

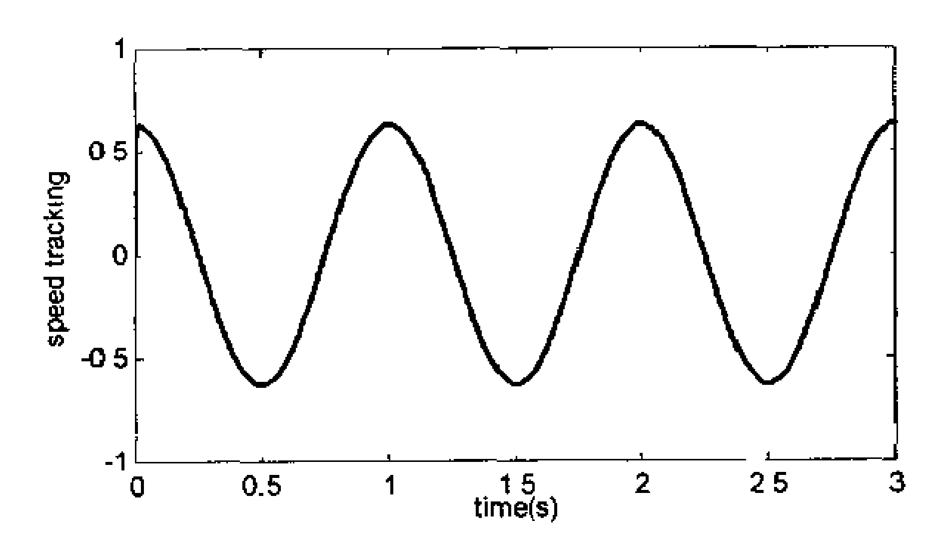


图 9-12 无摩擦时的速度跟踪

当 $M=1$ 时为带有摩擦环节的 PID 控制，仿真结果如图 9-13～图 9-16 所示。仿真结果表明在带有摩擦条件下，位置跟踪存在“平顶”现象，速度跟踪存在“死区”现象。采用 PID 控制鲁棒性差，不能达到高精度跟踪。

针对仿真方法二，仿真程序如图 9-17 所示。带有摩擦环节的 PID 控制仿真结果如图 9-18 和图 9-19 所示。仿真结果表明在带有摩擦条件下，位置跟踪存在“平顶”现象，速度跟踪存

在“死区”现象。采用PID控制鲁棒性差，不能达到高精度跟踪。

通过对两种仿真方法进行比较，可知采用基于S函数的Simulink仿真具有编程简单、运行速度快、对中间变量作图方便的特点。

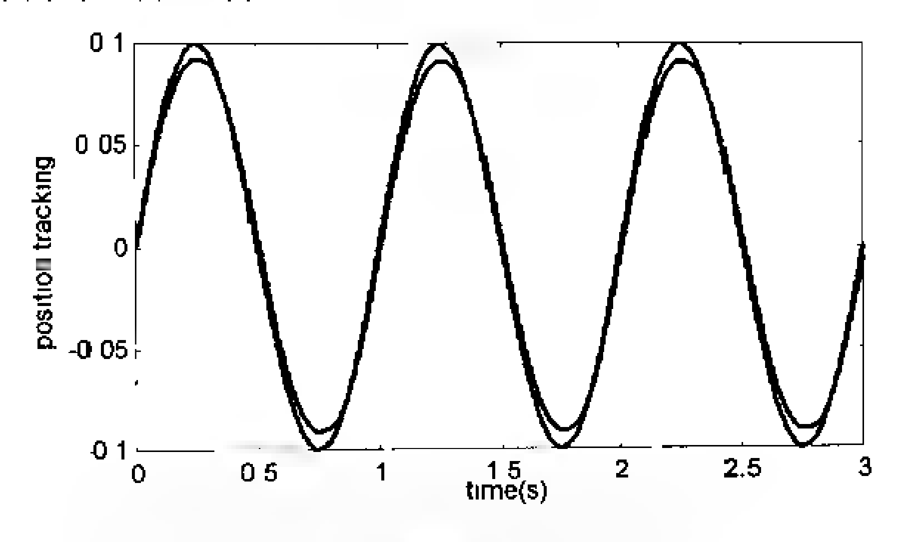


图9-13　带摩擦时的位置跟踪（M=1）

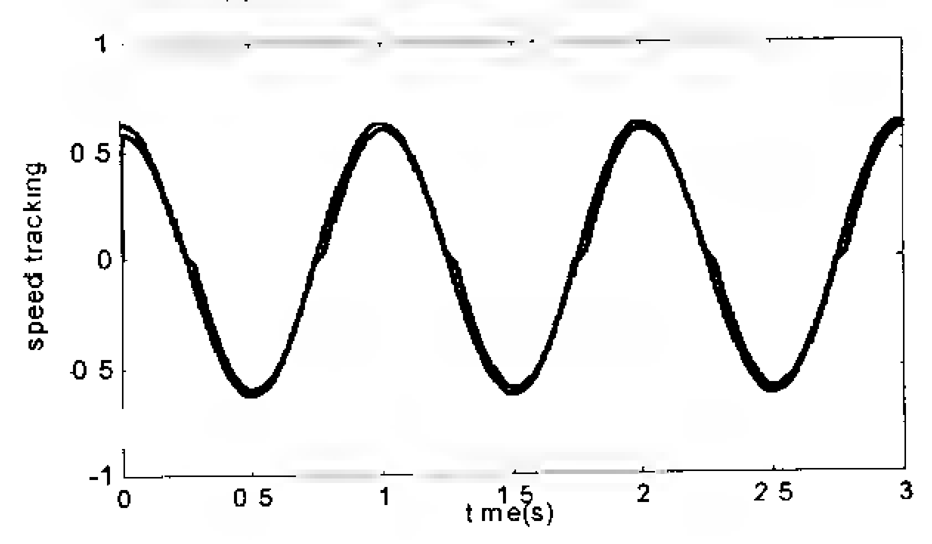


图9-14　带摩擦时的速度跟踪

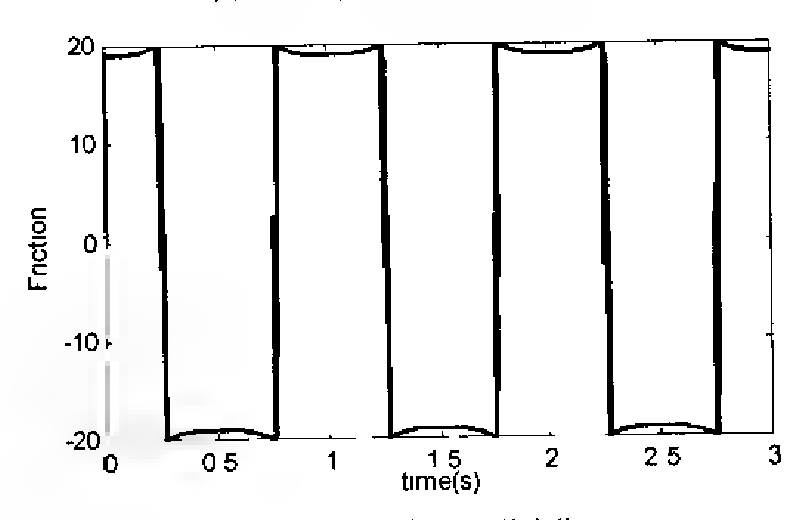


图9-15　摩擦力$F_f(t)$的变化

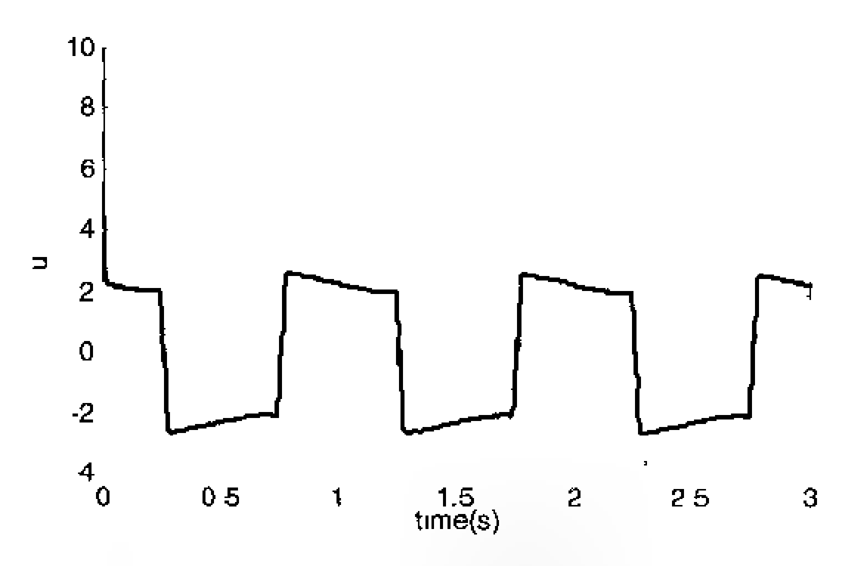


图9.16　带摩擦时控制器的输出

仿真方法一：采用M语言仿真

仿真程序：chap9_3.m。

```
%PID Control with Stribeck Friction Model
clear all;
close all;
global w A alfa J Ce R Km Ku S a1 Fm Fc M kv

%Servo system Parameters
J=0.6;Ce=1.2;Km=6;
Ku=11;R=7.77;

w=1*2*pi;A=0.10;

alfa=0.01;

T=3.0;
ts=0.001;  %Sampling time
TimeSet=[0:ts:T];

M=0;       %If M=0, No Friction works
S=1;
[t,x]=ode45('chap9_3f',TimeSet,[0,0,0]);
x1=x(:,1);
x2=x(:,2);
x3=x(:,3);

if S==1
   rin=A*sin(w*t);
   drin=A*w*cos(w*t);
```

```
   ddrin=-A*w*w*sin(w*t);
   error=rin-x(:,1);
   derror=drin-x(:,2);
end
if S==2
  for kk=1:1:T/ts+1
     rin(kk)=1;
     drin(kk)=0;
     ddrin(kk)=0;
     error(kk)=rin(kk)-x1(kk);
     derror(kk)=drin(kk)-x2(kk);
  end
end
if S==3
  for kk=1:1:T/ts+1
     rin(kk)=A*sign(sin(0.4*2*pi*t(kk)));
     drin(kk)=0;
     ddrin(kk)=0;
     error(kk)=rin(kk)-x1(kk);
     derror(kk)=drin(kk)-x2(kk);
  end
end

F=J*x(:,3);
x2=x(:,2);
for kk=1:1:T/ts+1
   time(kk)=(kk-1)*ts;

if abs(x2(kk))<=alfa
   if F(kk)>Fm
      Ff(kk)=Fm;
   elseif F(kk)<-Fm
      Ff(kk)=-Fm;
   else
      Ff(kk)=F(kk);
   end
end

if x2(kk)>alfa
   Ff(kk)=Fc+(Fm-Fc)*exp(-a1*x2(kk))+kv*x2(kk);
elseif x2(kk)<-alfa
   Ff(kk)=-Fc-(Fm-Fc)*exp(a1*x2(kk))+kv*x2(kk);
end
```

```
if M==0
   Ff(kk)=0;  %No Friction
end

u(kk)=200*error(kk)+40*derror(kk);   %PID Control

if u(kk)>=10
   u(kk)=10;
end
if u(kk)<=-10
   u(kk)=-10;
end
end
figure(1);
plot(t,rin,'k',t,x(:,1),'k');
xlabel('time(s)');ylabel('position tracking');
figure(2);
plot(t,drin,'k',t,x(:,2),'k');
xlabel('time(s)');ylabel('speed tracking');

figure(3);
plot(t,error,'k');
xlabel('time(s)');ylabel('error');
figure(4);
plot(x(:,2),Ff,'k');
xlabel('speed');ylabel('Friction');
figure(5);
plot(t,Ff,'k');
xlabel('time(s)');ylabel('Friction');
figure(6);
plot(time,u,'k');
xlabel('time(s)');ylabel('u');
```

子程序：chap9_3f.m。

```
function dx=Model(t,x)
global w A alfa J Ce R Km Ku S a1 Fm Fc M kv
persistent aa
dx=zeros(3,1);

a1=1.0;  %Effect on the shape of friction curve
Fm=20;
Fc=15;
```

```
kv 2.,;

F J*x( 3 ;
if t 0
   d.-0,
end
dF .*dd;

if abs(x(2  < alfa
   if F>Fm
      Ff Fm,
      dFf-0;
   elseif F< Fm
      Ff  Fm;
      dFf-0;
   else
      Ff F,
      dFf dF;
   end
end
 f x(2 >alfa
   Ff Fc+ Fm Fc *exp  a1*x(2, +kv*x(2);
   dFf- Fm Fc)*exp(-a1*x 2  *( a1 *x 3 +kv*x(3 ;
elseif x(2,< alfa
   Ff--Fc (Fm Fc *exp(a1*x 2) +kv*x(2);
   dFf   Fm Fc)*exp(a1*x 2),*a1*x 3)+kv*x 3);
end

if S 1
   rin A*sin w*t ;
   drin A*w*cos(w*t ;
   ddrin  A*w*w*sin w*t);
   dddrin  A*w*w*w*cos(w*t);
end
if S 2
   rin 1;
   drin 0;
   ddrin 0;
   dddrin 0;
end
if S 3
   rin A*sign(sin(0.4*2*pi*t));
   drin 0;
```

```
   ddrin=0;
   dddrin=0;
end
error=rin-x(1);
derror=drin-x(2);
dderror=ddrin-x(3);

u=200*error+40*derror;   %PID
du=200*derror+40*dderror;

if u>=110
   u=110;
end
if u<=-110
   u=-110;
end

if M==0
   Ff=0;dFf=0;   %No Friction
end
dx(1)=x(2);
dx(2)=-Km*Ce/(J*R)*x(2)+Ki*Km*u/(J*R)-Ff/J;
dx(3)=-Km*Ce/(J*R)*x(3)+Ki*Km*du/(J*R)-dFf/J;
sys=dx;
```

仿真方法二：基于 *S* 函数的 Simulink 仿真

采用 *S* 函数实现对象及摩擦模型的表示及位置、速度和摩擦力的输出。仿真结果如图 9-17 和图 9-18 所示。

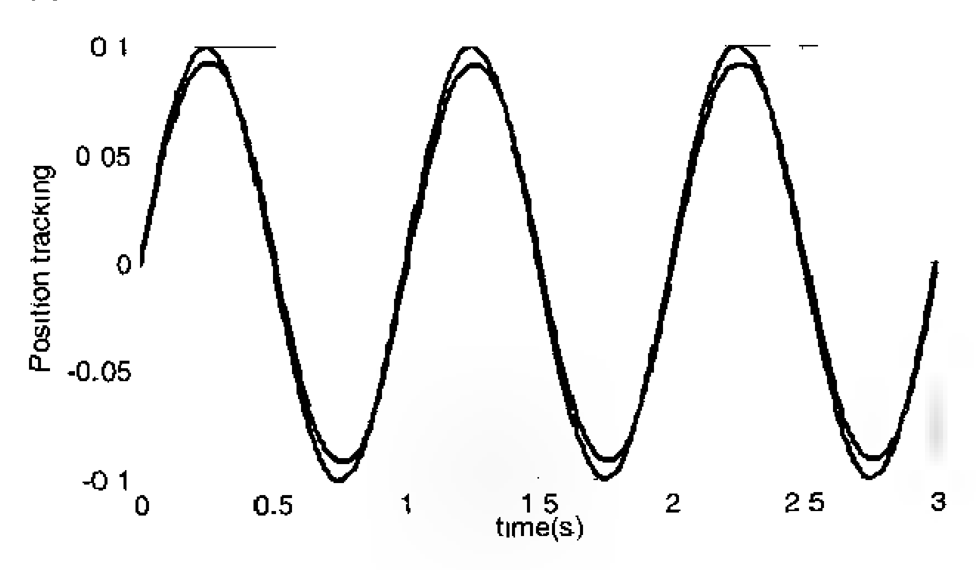


图 9-17　带摩擦时的位置跟踪

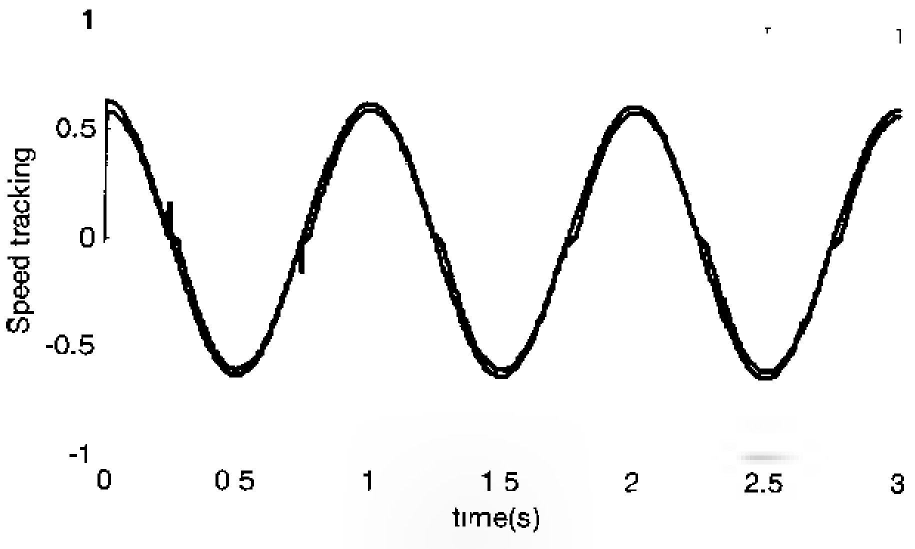


图 9-18　带摩擦时的速度跟踪

仿真程序：chap9_4.mdl，如图 9-19 所示。

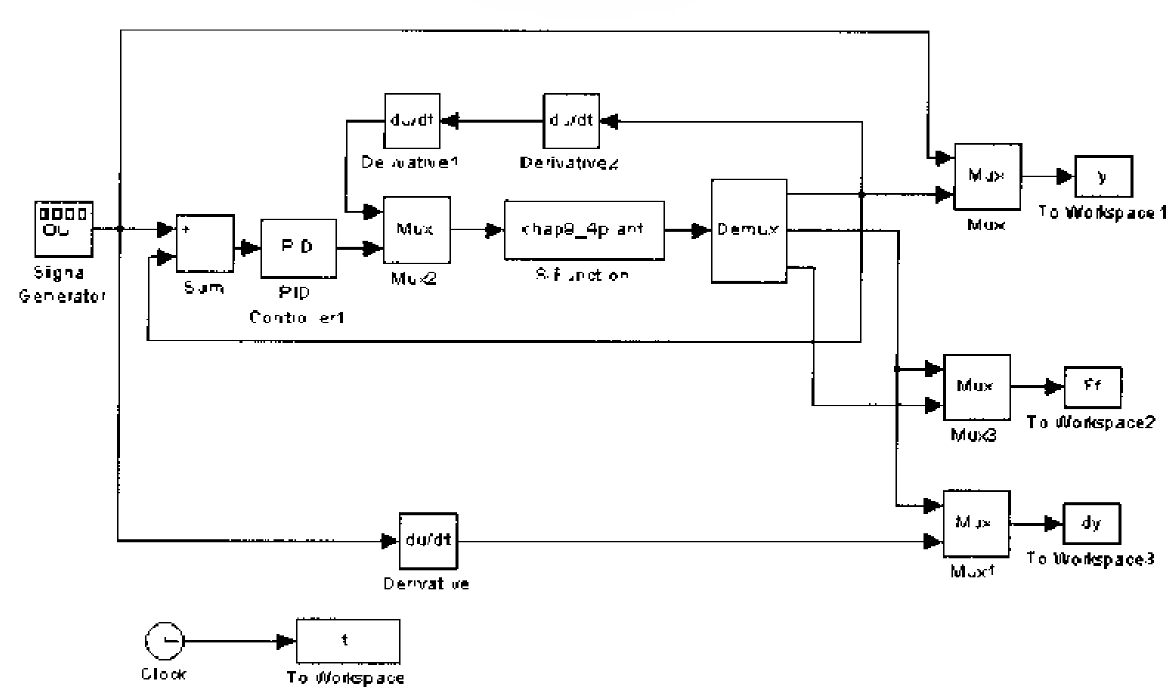


图 9-19　Simulink 仿真程序

被控对象 *S* 函数子程序：chap9_4plant.m

```
function [sys,x0,str,ts]=spacemodel(t,x,u,flag)

switch flag,
case 0,
    sys,x0,str,ts]=mdlInitializeSizes;
case 1,
   sys=mdlDerivatives(t,x,u);
case 3,
   sys=mdlOutputs(t,x,u);
```

```
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end

function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 2;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 3;
sizes.NumInputs      = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1; % At least one sample time is needed
sys = simsizes(sizes);
x0  = [0;0];
str = [];
ts  = [0 0];

function sys=mdlDerivatives(t,x,u)

%Servo system Parameters
J=0.6;Ce=1.2;Km=6;
Ku=11;R=7.77;
kv=2.0;

alfa=0.01;
a1=1.0;  %Effect on the shape of friction curve
Fm=20;
Fc=15;
kv=2.0;

F=J*u(1);
if abs(x(2))<=alfa
   if F>Fm
      Ff=Fm;
   elseif F<-Fm
      Ff=-Fm;
   else
      Ff=F;
   end
end
```

```
if x(2)>alfa
   Ff=Fc+(Fm-Fc)*exp(-a1*x(2))+kv*x(2);
elseif x(2)<-alfa
   Ff=-Fc-(Fm-Fc)*exp(a1*x(2))+kv*x(2);
end

sys(1)=x(2);
sys(2)=-Km*Ce/(J*R)*x(2)+Ku*Km*u(1)/(J*R)-Ff/J;

function sys=mdlOutputs(t,x,u)

%Servo system Parameters
J=0.6;Ce=1.2;Km=6;
Ku=11;R=7.77;
kv=2.0;

alfa=0.01;
a1=1.0;  %Effect on the shape of friction curve
Fm=20;
Fc=15;
kv=2.0;

F=J*u(1);
if abs(x(2))<=alfa
   if F>Fm
      Ff=Fm;
   elseif F<-Fm
      Ff=-Fm;
   else
      Ff=F;
   end
end
if x(2)>alfa
   Ff=Fc+(Fm-Fc)*exp(-a1*x(2))+kv*x(2);
elseif x(2)<-alfa
   Ff=-Fc-(Fm-Fc)*exp(a1*x(2))+kv*x(2);
end

sys(1)=x(1);    %Angle
sys(2)=x(2);    %Angle speed
sys(3)=Ff;      %Friction force
```

作图程序：chap9_4plot.m。

```
close all;
figure(1);
plot(t,y(:,1),'k',t,y(:,2),'k');
xlabel('time(s)');ylabel('Position tracking');

figure(2);
plot(Ff(:,1),Ff(:,2),'k');
xlabel('Angle speed');ylabel('Friction force');

figure(3);
plot(t,dy(:,1),'k',t,dy(:,2),'k');
xlabel('time(s)');ylabel('Speed tracking');
```

9.3 伺服系统三环的 PID 控制

9.3.1 伺服系统三环的 PID 控制原理

以转台伺服系统为例，其控制结构如图 9-20 所示，其中 r 为框架参考角位置输入信号，θ 为输出角位置信号。

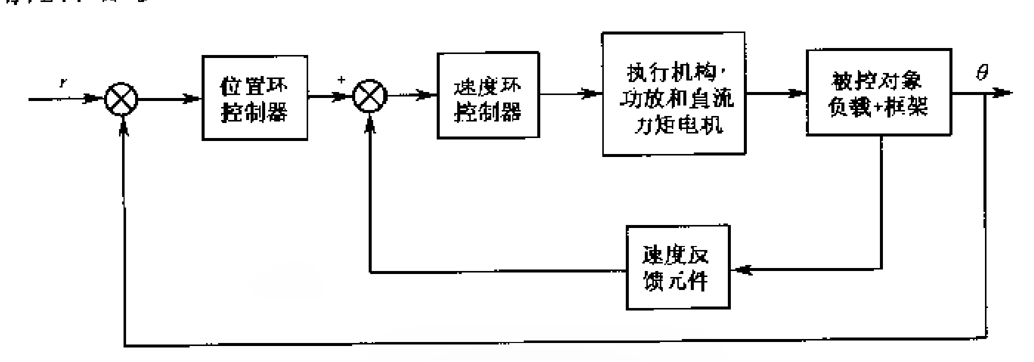


图 9-20 转台伺服系统框图

伺服系统执行机构为典型的直流电动驱动机构，电机输出轴直接与负载-转动轴相连，为使系统具有较好的速度和加速度性能，引入测速机信号作为系统的速度反馈，直接构成模拟式速度回路。由高精度圆感应同步器与数字变换装置构成数字式角位置伺服回路。

转台伺服系统单框的位置环、速度环和电流环框图如图 9-21、图 9-22 和图 9-23 所示。

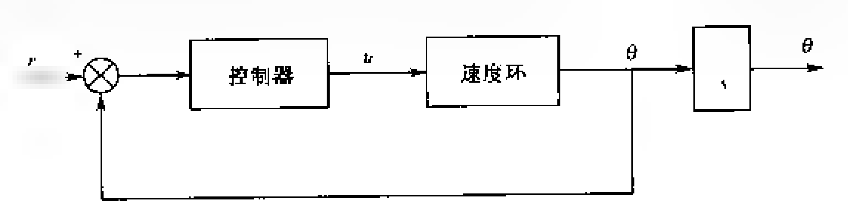


图 9-21 伺服系统位置环框图

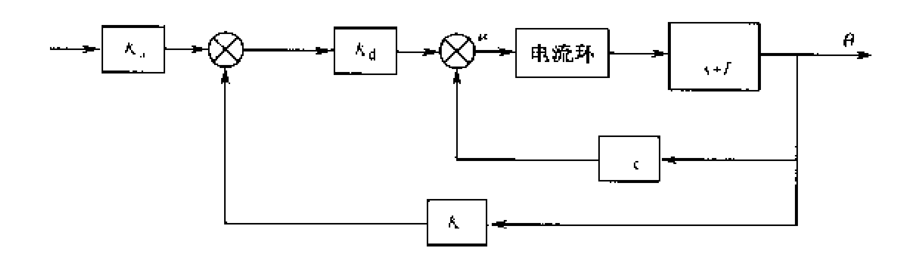


图 9-22 伺服系统速度环框图

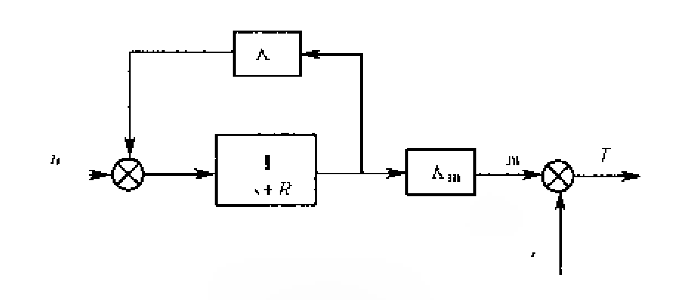


图 9-23 伺服系统电流环框图

图中符号含义如下：r 为位置指令；θ 为转台转角；K_u 为 PWM 功率放大倍数；K_d 为速度环放大倍数；K_v 为速度环反馈系数；K_i 为电流反馈系数；L 为电枢电感；R 为电枢电阻；K_m 为电机力矩系数；C_e 为电机反电动势系数；J 为等效到转轴上的转动惯量；b 为粘性阻尼系数，其中 $J=J_m+J_L$，$b=b_m+b_L$，J_m 和 J_L 分别为电机和负载的转动惯量，b_m 和 b_L 分别为电机和负载的粘性阻尼系数；T_f 为扰动力矩，包括摩擦力矩和耦合力矩。

假设在速度环中的外加干扰为粘性摩擦模型：

$$F_f(t)=F_c\ \mathrm{sgn}(\dot{\theta})+b_c\ \dot{\theta} \tag{9.9}$$

控制器采用 PID 控制+前馈控制的形式，加入前馈摩擦补偿控制表示为：

$$u_f(t)=F_{c1}\ \mathrm{sgn}(\dot{\theta})+b_{c1}\ \dot{\theta} \tag{9.10}$$

式中，F_{c1} 和 b_{c1} 为粘性摩擦模型等效到位置环的估计系数，该系数可以根据经验确定，或根据计算得出。

9.3.2 仿真程序及分析

仿真实例

被控对象为一个具有三环结构的伺服系统。伺服系统参数和控制参数在程序中给出描述，系统采样时间为 1ms。取 $M=2$，此时输入指令为正弦叠加信号：$r(t)=A\sin(2\pi Ft)+0.5A\sin(1.0\pi Ft)+0.25A\sin(0.5\pi Ft)$，其中 $A=0.50$，$F=0.50$。

考虑到 K_i，L 和 C_e 的值很小，前馈补偿系数 F_{c1} 和 b_{c1} 等效到摩擦力矩端的系数可近似写为：

$$\mathrm{Gain}\approx K_u\times K_d\times 1/R\times K_m\times K_g$$

式中，K_g 为经验系数，摩擦模型估计系数 F_{c1} 和 b_{c1} 为：

$$F_{c1}\approx F_c/\mathrm{Gain}$$

$$b_{c1}\approx b_c/\mathrm{Gain}$$

系统总的控制输出为：

$$u(t)=u_p(t)+u_f(t)$$

式中，$u_p(t)$ 为 PID 控制的输出，其三项系数为 $k_{pp}=15$，$k_{ii}=0.10$，$k_{dd}=1.5$。

根据是否加入摩擦干扰和前馈补偿分别进行仿真。

初始化程序：chap9_5i.m。

```
%Three Loop of Flight Simulator Servo System with Direct Current Motor
clear all;
close all;
% 1)Current loop
L=0.001;   %L<<1 Inductance of motor armature
R=1;       %Resistence of motor armature
ki=0.001;  %Current feedback coefficient

%(2)Velocity loop
kd=6;      %Velocity loop amplifier coefficient
kv=2;      %Velocity loop feedback coefficient

J=2;       %Equivalent moment of inertia of frame and motor
b=1;       %Viscosity damp coefficient of frame and motor

km=1.0;    %Motor moment coefficient
Ce=0.001;  %Voltage feedback coefficient

%Friction model: Coulomb&Viscous Friction
Fc=100.0;bc=30.0;  %Practical friction

%(3)Position loop: PID controller
ku=11;     %Voltage amplifier coefficient of PWM
kpp=150;
kii=0.1;
kdd=1.5;

%Friction Model compensation
%Equavalent gain from feedforward to practical friction
Gain=ku*kd*1/R*km*1.0;
Fc1=Fc/Gain;    bc1=bc/Gain; %Feedforward compensation

%Input signal initialize
F=0.50;
A=0.50;
ts=0.001;   %Sampling time
```

```
M 2;
.I M    .         %^ine ^ gnal
    k ^     ;
    .ime     .t.:k* <] ;   %Simulatio    me
    rin A*s.n ^* 1*F* .me ;
    drin   *p1*F*A*   s    *p.*F*t.me ;
elseif M  .   %Rand m Signal
    T 4..9
    time zeros  .,1 ;
    r.n zer s T,. ,
    drin zeros .,1 ;
    .1..  .    ;
    dr1..  .    ,
for k 1:1 T
    time k+.  k*t ;
%Rand m signal
  r1. k+1) A*sin(2*pi*F*k*ts)+(.5*A*sin .*p.*0 5*F*k*ts +..
          ^..5*A*sin(2*p.*0.25*F*k*ts ;
  drin k+1   rin k+1  r.n k)) ts,
end
end
```

控制系统的 Simulink 程序：chap9_5.mdl，如图 9-24 和图 9 25 所示。

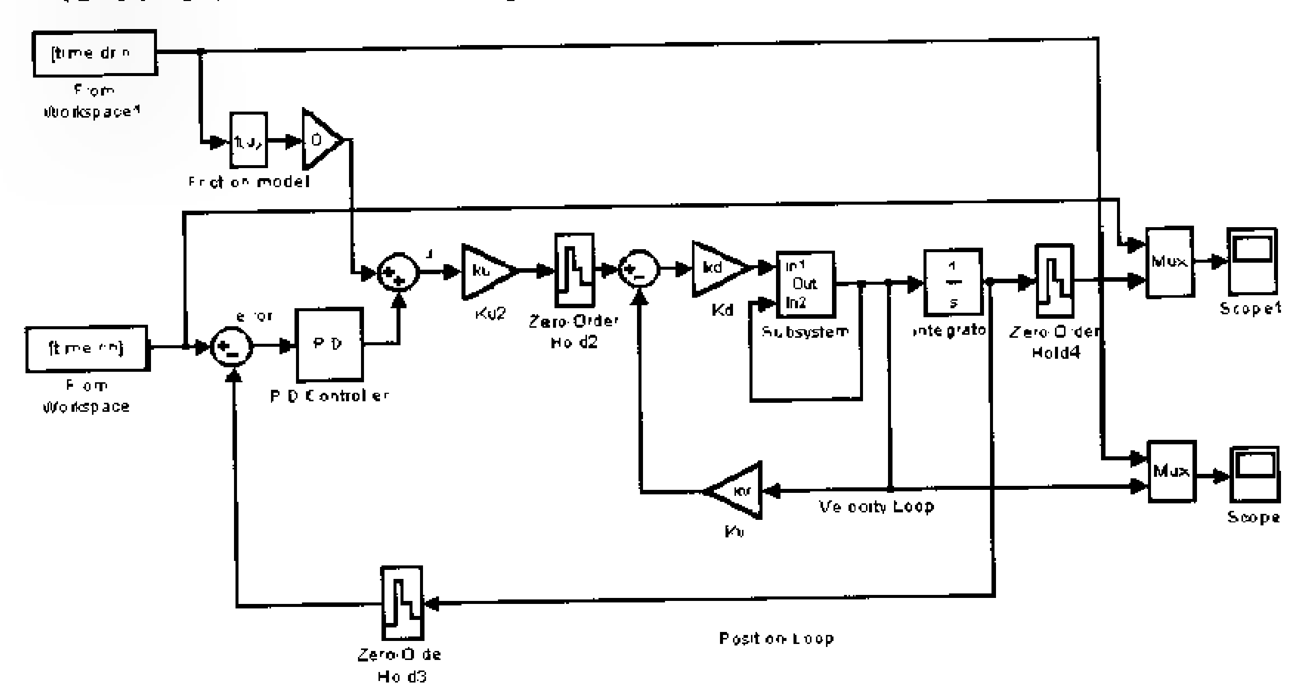


图 9 24　　环控制的 Simulink 仿真程序

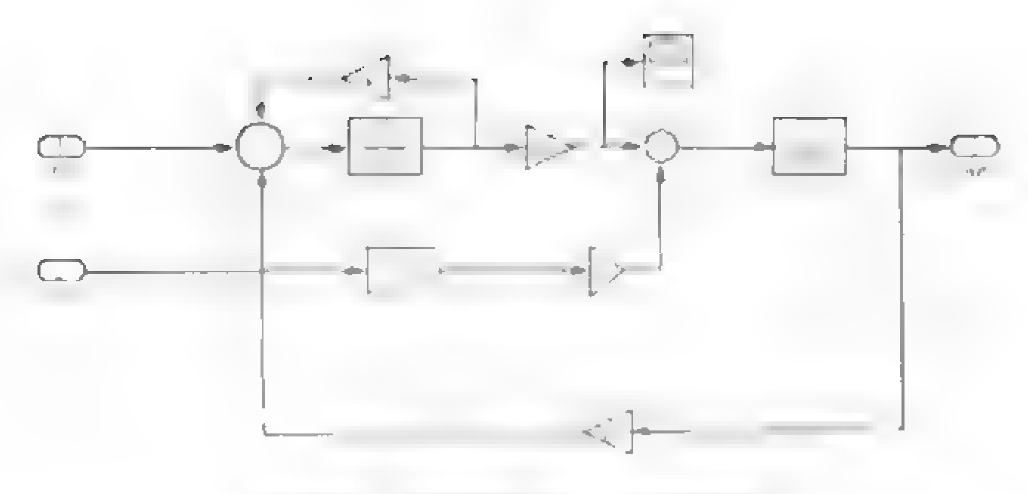


图 9-25 [illegible] Simulink [illegible]

1. 无摩擦无[illegible]，[illegible]如图 9-26 和图 9-27 所示

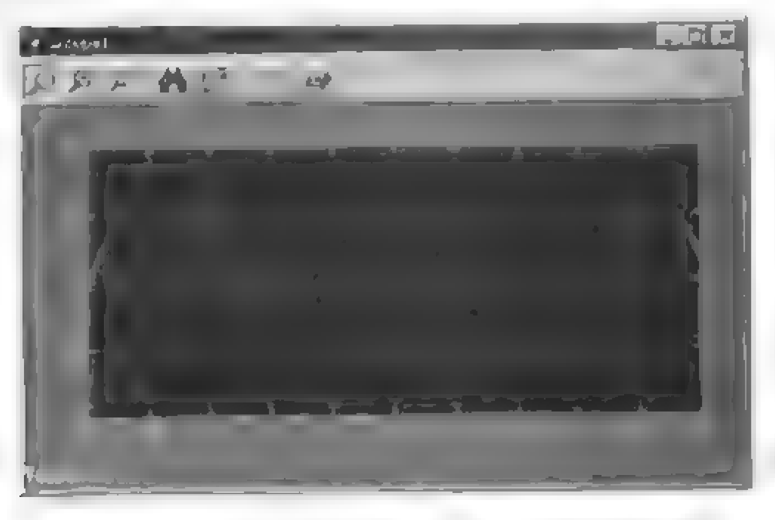


图 9-26 [illegible]（无摩擦[illegible]）

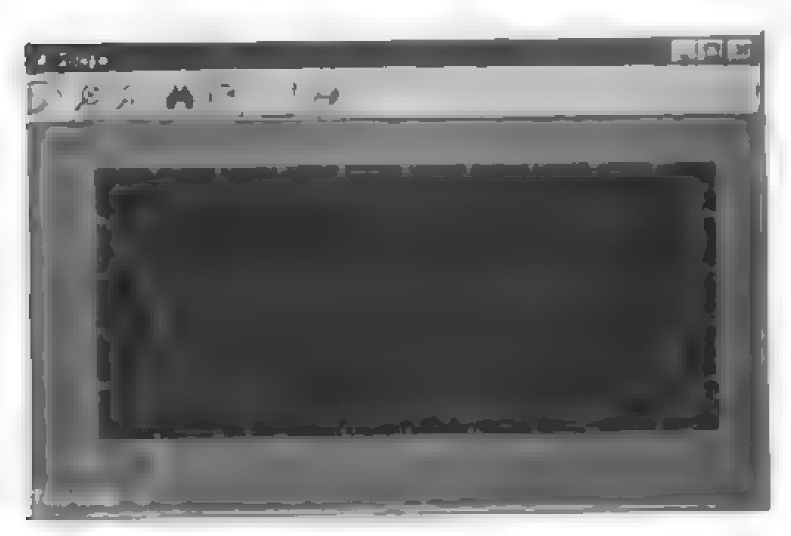


图 9-27 正弦叠加信号速度跟踪（无摩擦无前馈补偿）

2. 带摩擦无[illegible]时的仿真，正弦叠加信号跟踪如图 9-28 和图 9-29 所示。由于静摩擦的作用，[illegible]位置跟踪存在“平顶”现象，速度跟踪存在“死区”现象。

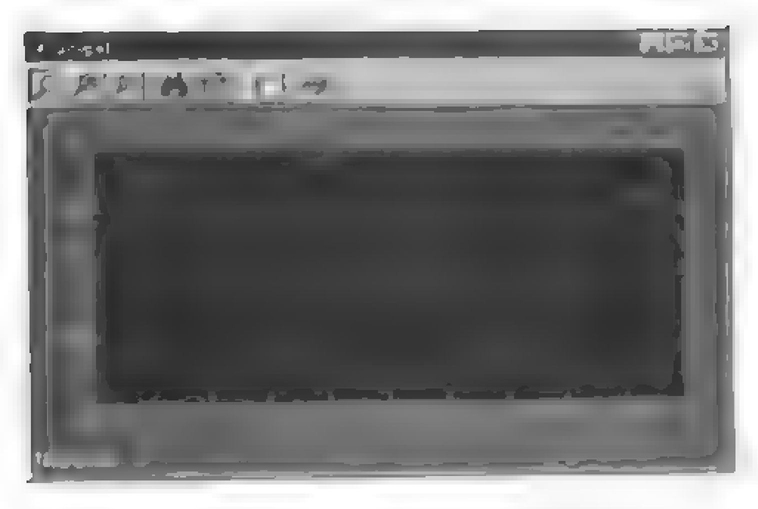


图 9-28　正弦叠加信号位置跟踪（带摩擦[illegible]补偿）

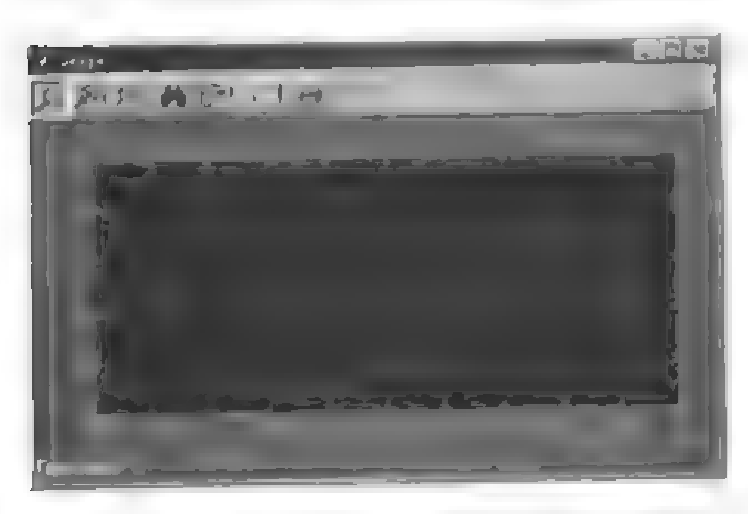


图 9-29　正弦[illegible]速度跟踪（带摩擦[illegible]补偿）

3）带摩擦在前馈补偿控制[illegible]如图 9-30 和图 9-31 所示。采用 PID 控制加前馈控制[illegible]了位置跟踪的"平顶"和速度跟踪的"死区"，实现了较高的[illegible]

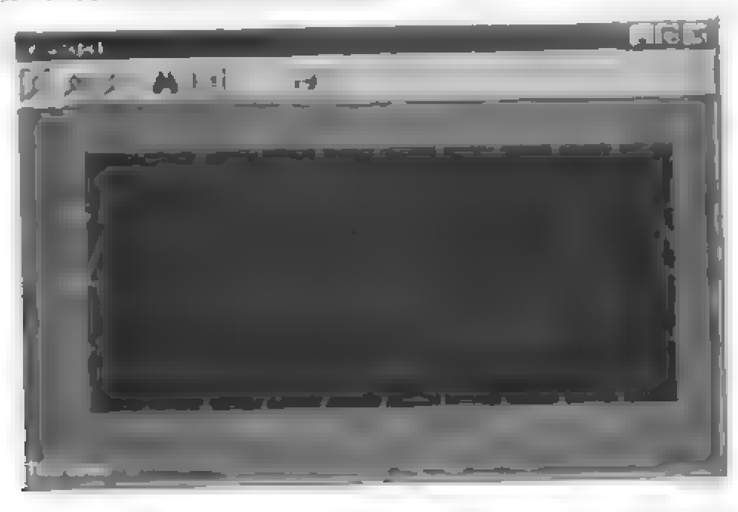


图 9-30　[illegible]

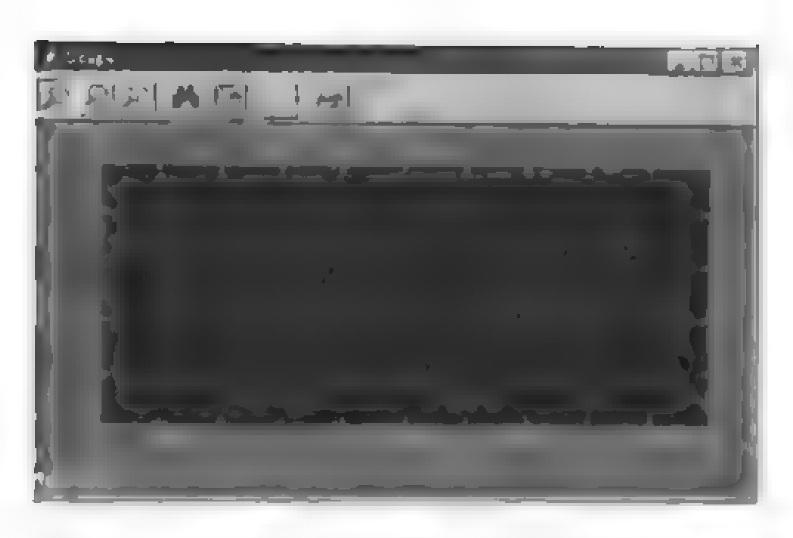


图 9-31　正弦叠加信号跟踪结果（带摩擦自适应补偿）

9.4　二质量伺服系统的 PID 控制

9.4.1　二质量伺服系统的 PID 控制原理

如果伺服系统把电机与负载作为一个整体来考虑，则称为单质量伺服系统。该系统与实际特性有很大差别。对于实际系统，尽管电机与负载是直接耦合的，但传动本质上是弹性的，而且轴承和框架也都不完全是刚性的。在电机驱动力矩的作用下，机械轴会受到某种程度的扭曲和变形。对于加速要求大、快速性和精度要求高的系统或是转动惯量大、性能要求高的系统，弹性变形对系统性能的影响不可忽略。由于传动轴的弯曲和变形，在传递运动时会存储能量。如果阻尼很小，则在宽范围内将产生较高的机械谐振。此谐振对系统的动态性能影响较大。对此可将模型等效为如图 9-32 所示的由电机、弹性负载及连接二者的等效传动轴所组成的二质量系统。

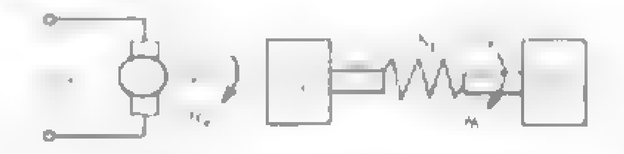


图 9-32　电机-传动轴-负载模型

根据图 9-32 中的传动轴动力学方程，根据伺服系统电机框图（如图 9-33 所示），可得电机动力学方程。根据伺服系统负载框图（如图 9-34 所示），不考虑干扰时，可得负载动力学方程

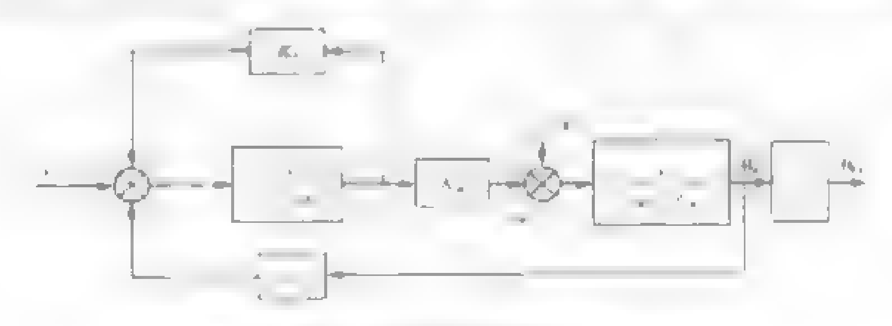


图 9-33　伺服系统电机框图

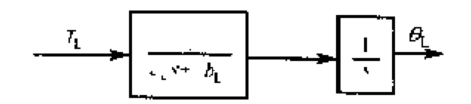


图 9-34　伺服系统负载框图

三质量伺服系统的电学方程和动力学方程为：

电机

$$iR + Li = u_a - C_e\dot{\theta}_m - K_t i \tag{9.11}$$

$$T_m = iK_m \tag{9.12}$$

$$J_m\ddot{\theta}_m = T_m - b_m\dot{\theta}_m - K_L(\theta_m - \theta_L) \tag{9.13}$$

传动轴

$$J_a(\ddot{\theta}_m - \ddot{\theta}_L) = K_L(\theta_m - \theta_L) - T_{mL} \tag{9.14}$$

负载

$$J_L\ddot{\theta}_L = T_{mL} - b_L\dot{\theta}_L \tag{9.15}$$

式中，J_a 为传动轴的转动惯量，θ_m 和 θ_L 分别为电机和负载的转角，J_m 和 J_L 分别为电机和负载的转动惯量，b_m 和 b_L 分别为电机和负载的粘性阻尼系数；K_L 为电机和框架之间的耦合刚度系数，T_{mL} 为负载端输出力矩。

一般 J_a 相对于 J_L 很小，而且其质量分布在轴的长度上，因此可以忽略或计入到 J_L 中，于是上述三质量系统可以简化为二质量系统。二质量系统的电学和动力学方程为：

电机

$$iR + Li = u_a - C_e\dot{\theta}_m - K_t i \tag{9.16}$$

$$T_m = iK_m \tag{9.17}$$

$$J_m\ddot{\theta}_m = T_m - b_m\dot{\theta}_m - K_L(\theta_m - \theta_L) \tag{9.18}$$

负载

$$J_L\ddot{\theta}_L = T_{mL} - b_L\dot{\theta}_L \tag{9.19}$$

$$K_L(\theta_m - \theta_L) - T_{mL} = 0 \tag{9.20}$$

根据上述描述，得到二质量伺服系统部分结构框图（其余部分与单质量伺服系统结构相同），如图 9-35 所示。

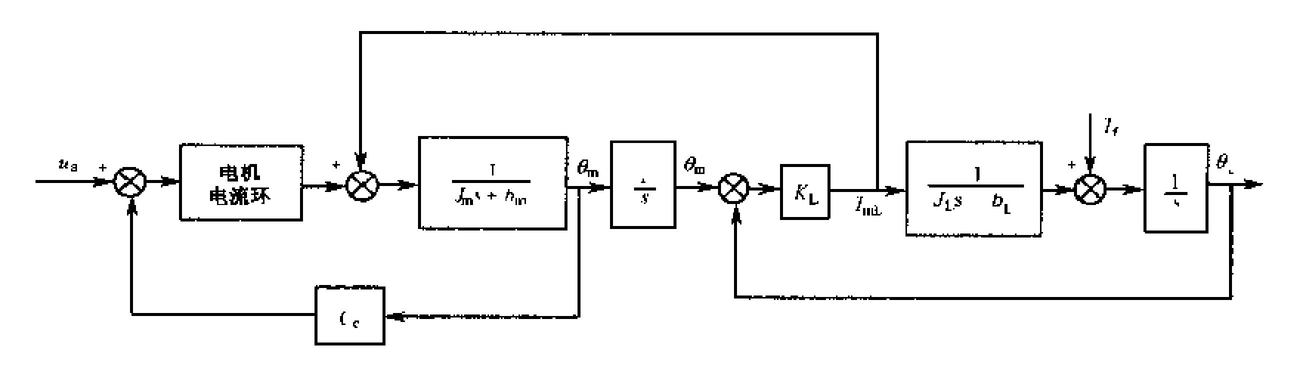


图 9-35　二质量伺服系统部分结构框图

9.4.2　仿真程序及分析

仿真实例

被控对象为一个具有三环结构的二质量伺服系统。伺服系统参数和控制参数在程序中给出描述，系统采样时间为 1ms。

输入指令为正弦信号：$r(t)=A\sin(2\pi Ft)$，其中 $A=0.50$，$F=0.50$。

假设在速度环中的外加干扰 T_f 为粘性摩擦模型：$F_f(t)=F_c\cdot\mathrm{sgn}(\theta)+b_c\cdot\dot{\theta}$。

控制器采用 PID 控制，参数选为：$k_{pp}=8.0$，$k_{ii}=1.0$，$k_{dd}=5.0$。

根据是否加入摩擦干扰分别进行仿真。

初始化程序：chap9_6i.m。

```
%Three Loop of Flight Simulator Servo System with two mass of Direct Current Motor
clear all;
close all;

%(1)Current loop
L=0.001;      %L<<1,Inductance of motor armature
R=1.0;        %Resistence of motor armature
ki=0.001;     %Current feedback coefficient

%(2)Velocity loop
kd=6;         %Velocity loop amplifier coefficient
kv=2;         %Velocity loop feedback coefficient

Jm=0.005;     %Equivalent moment of inertia of motor
bm=0.010;     %Viscosity damp coefficient of motor

km=10;        %Motor moment coefficient
Ce=0.001;     %Voltage feedback coefficient

Jl=0.15;      %Equivalent moment of inertia of frame
bl=8.0;       %Viscosity damp coefficient of frame

kl=5.0;       %Motor moment coefficient between frame and motor

%Friction model: Coulomb&Viscous Friction
Fc=10;bc=3;   %Practical friction

%(3)Position loop: PID controller
ku=11;            %Voltage amplifier coefficient of PWM

kpp=8;
kii=1.0;
kdd=5;
```

```
%Input Signal Initialize
F=50;
A=0.50;
ts=0.001;          %Sampling time

k=5000;
time=[0:ts:k*ts]';  %Simulation time

rin=A*sin(2*pi*F*time);
drin=2*pi*F*A*cos(2*pi*F*time);
```

控制系统的 Simulink 程序：chap9_6.mdl，如图 9-36～图 9-38 所示。

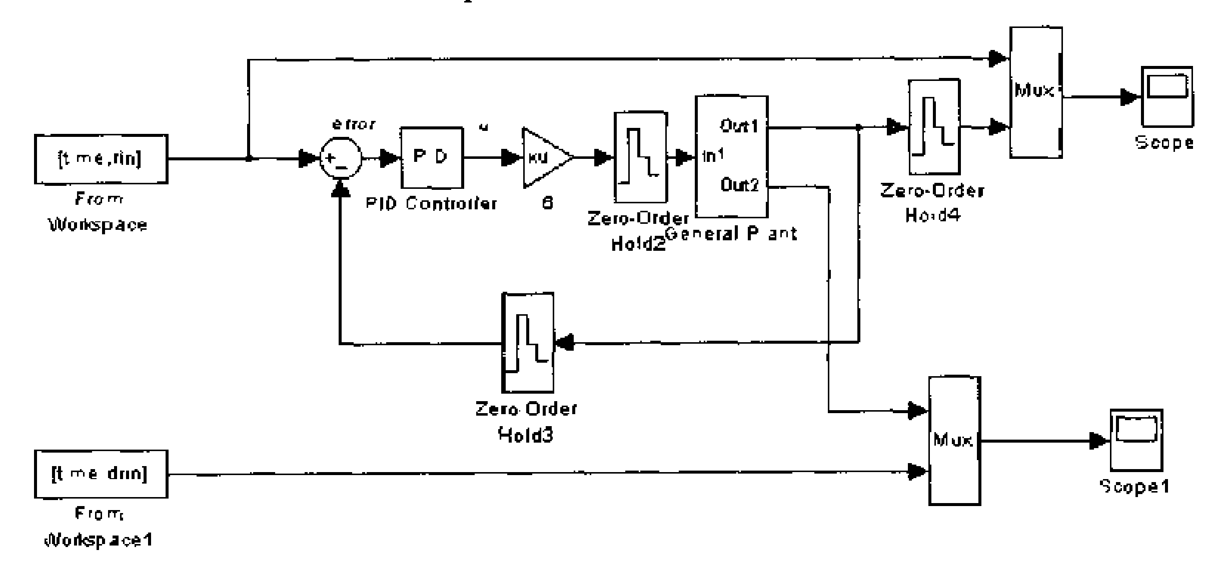


图 9-36 单质量伺服系统三环控制的 Simulink 仿真主程序

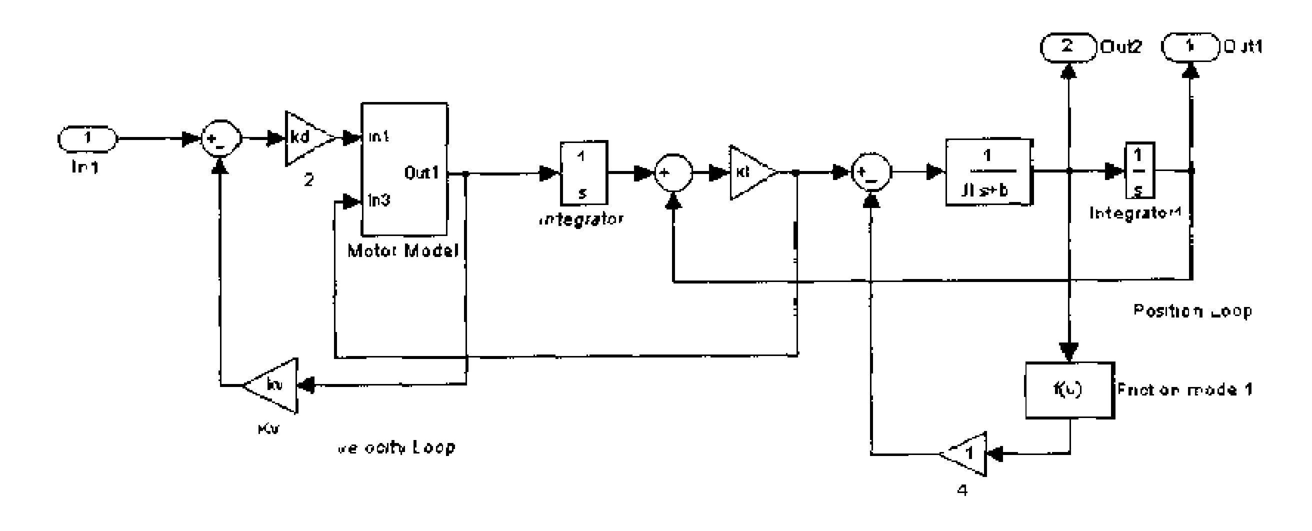
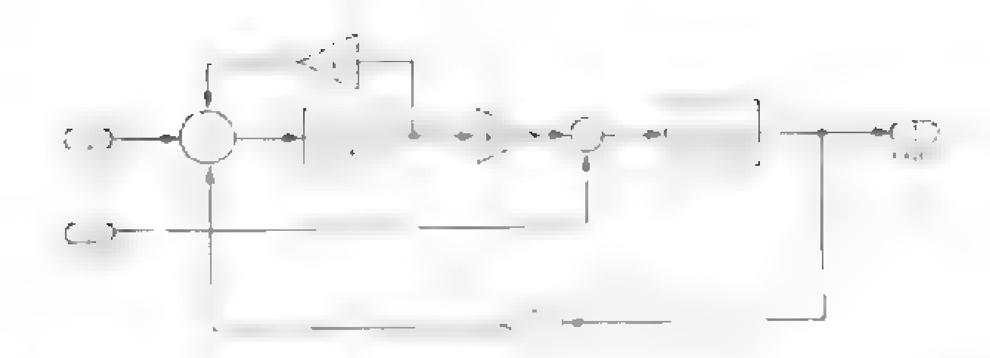
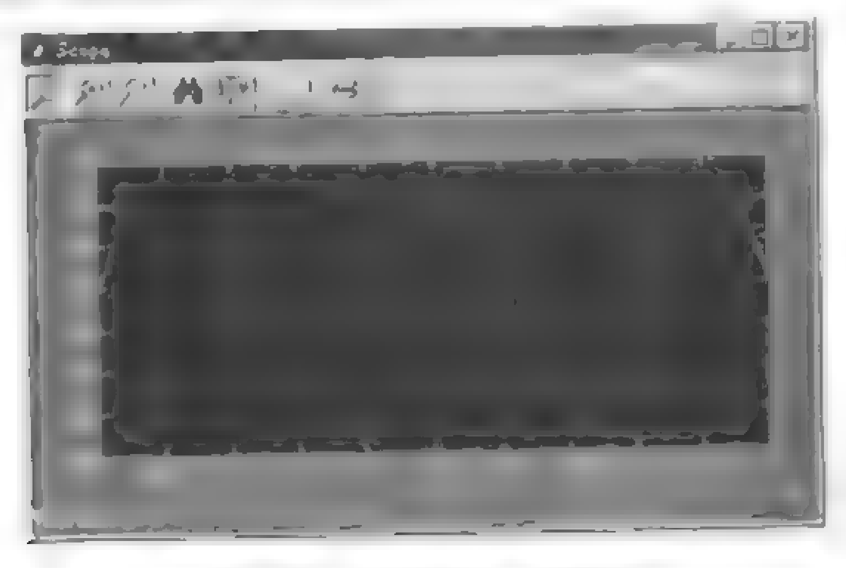


图 9-37 广义被控对象的 Simulink 仿真程序



[illegible] Simulink 仿真[illegible]

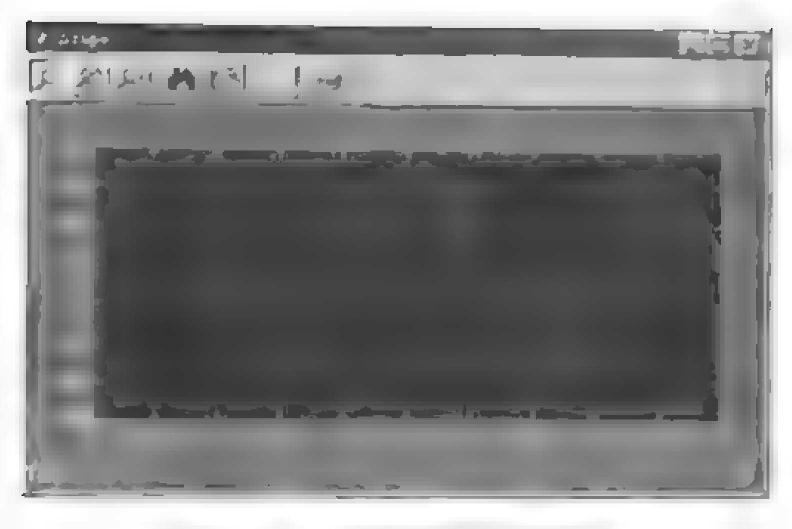
[illegible]的结果如图 9-39 和图 9-40 所示

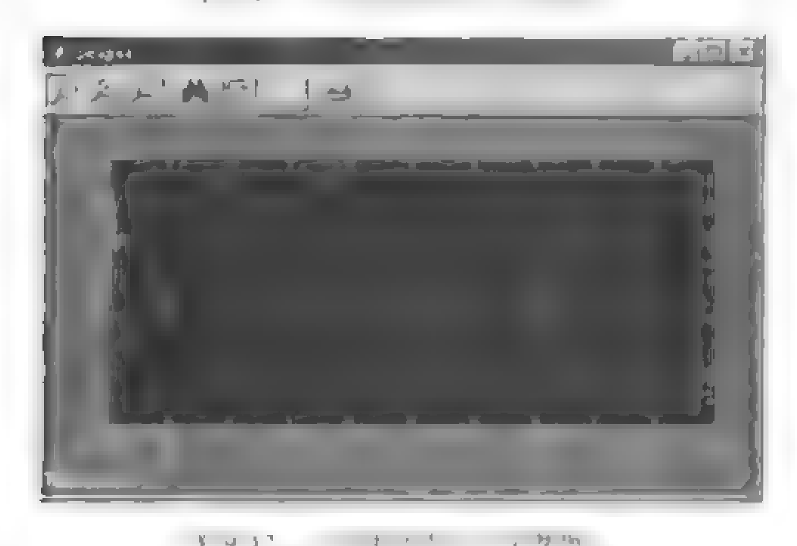


[illegible]

[illegible]速度跟踪[illegible]

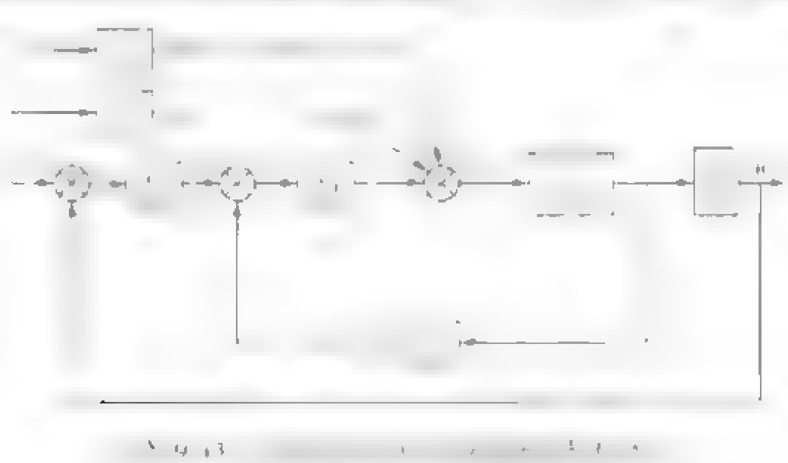
2 [illegible]位置跟踪[illegible]的结果如图 9-41 和图 9-42 所示。由于静摩擦[illegible]，[illegible]位置跟踪存在“[illegible]顶”现象，速度跟踪存在“[illegible]”现象





9.5 伺服系统的模拟PD+数字前馈控制

9.5.1 伺服系统的模拟PD+数字前馈控制原理



采用 PD 加前馈控制方式，设计的控制律如

$$u = k_d[k_p(r-\theta) - k_v\dot{\theta}] + f_1\dot{r} + f_2\ddot{r} = k_1 e - k_2\dot{\theta} + f_1\dot{r} + f_2\ddot{r} \tag{9.21}$$

式中，$k_1 = k_d k_p$，$k_2 = k_d k_v$，$e = r - \theta$

由图 9-43 中可知

$$\frac{1}{Js^2 + bs} = \frac{\theta}{u}$$

即

$$J\ddot{\theta} + b\dot{\theta} = u$$

[illegible] 式，得：

$$[illegible]$$

[illegible]

$$[illegible]$$

[illegible]

$$[illegible]$$

[illegible]

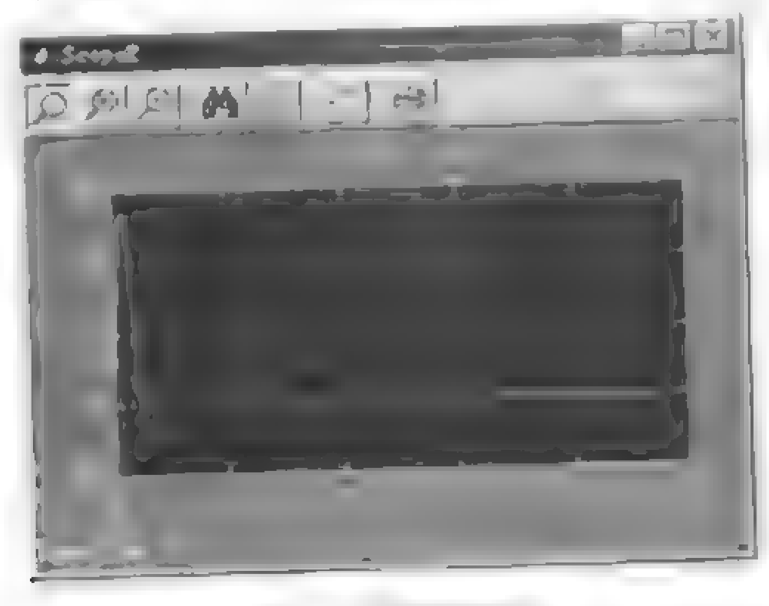
$$[illegible]$$

[illegible] 系统的跟踪误差 $e(t)$ 收敛于零

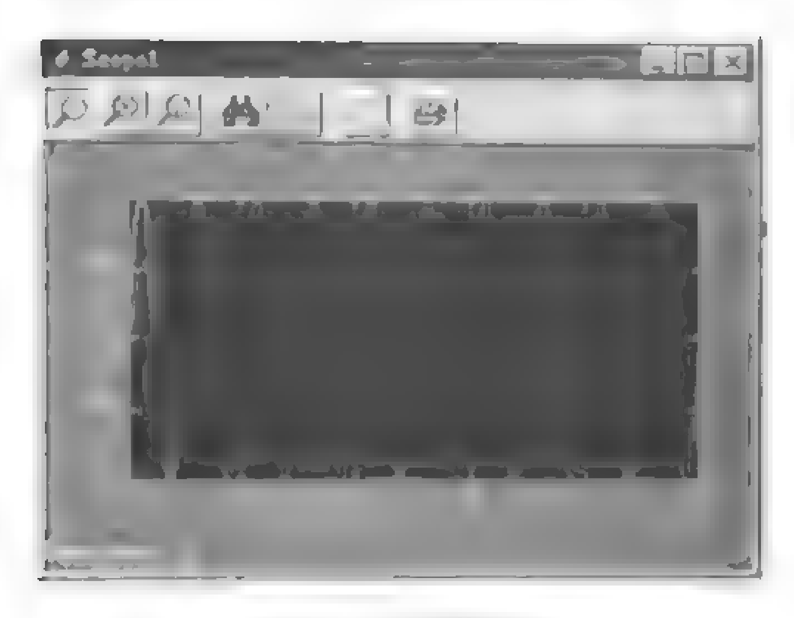
9.5.2 仿真程序及分析

仿真实例

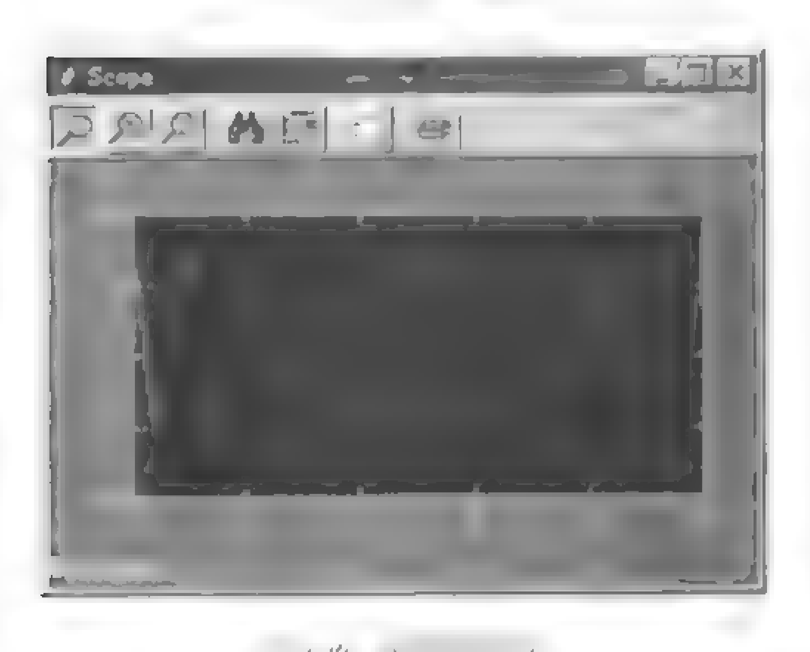
被控对象为一个具有 [illegible] 环结构的伺服 [illegible] 数和控制参数在程序中给出描 [illegible]。系统输入信号的采样时间为 1ms，输 [illegible] 号，$r(t) = A\sin(2\pi Ft)$，其中 [illegible] [illegible] $u(t)$ 为控制器 [illegible]，$J = 2.0\text{kg}\cdot\text{m}^2$，$b = 0.50$，$k_v = 2.0$，[illegible] $= 1.0$，$F = 1.0$，$k_p = 15$，$k_d = 6$，则 $f_1 = k_2 + b$，$f_2 = J$。仿真 [illegible] 图 9-44～图 9-46 所示。



[illegible]



9.45 [illegible]



[illegible]

初始化程序：chap9_7i.m

```
[illegible]

F=1;
A=1;
t=[0:0.001:10]';  %Simulation time
```

主程序：chap9_7.mdl，如图 9-47 所示。

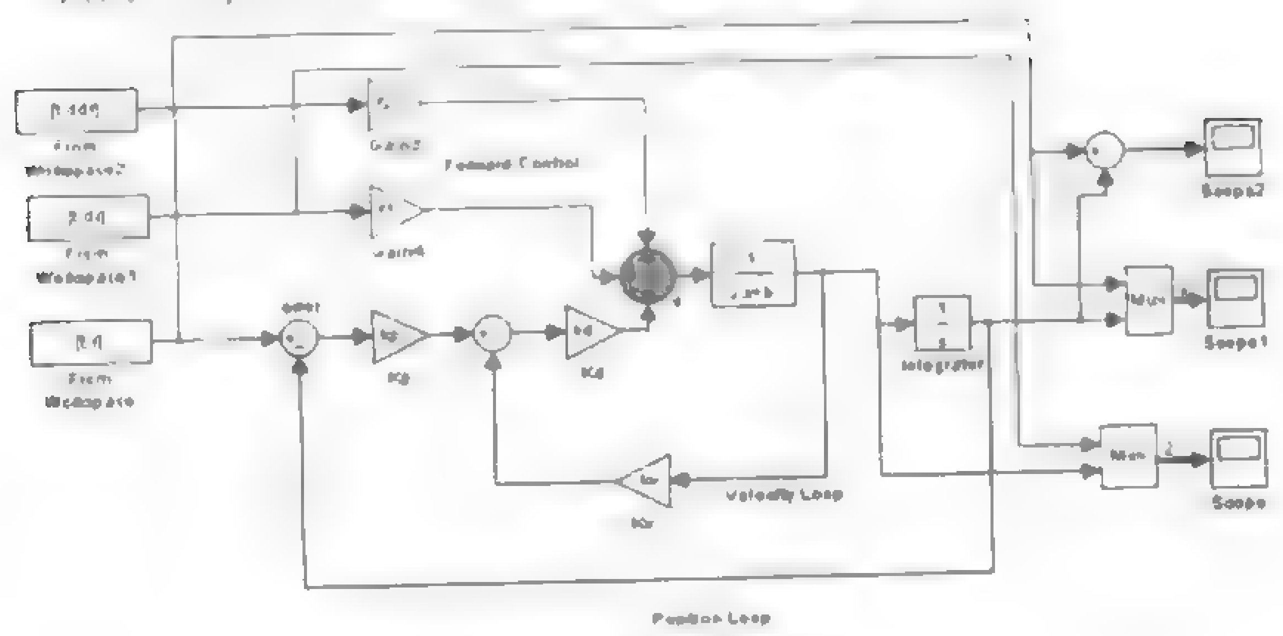


图 9-47 Simulink 主程序

第 10 章　机器人的 PID 控制

针对单臂机械手的机器人控制，申铁龙教授针对确定性和不确定性单臂机械手，分别提出了一种基于“PD+前馈”的控制方法和一种基于“PD+鲁棒”的鲁棒控制方法，并证明了这两种算法的稳定性[40]。下面分别介绍这两种控制算法，并通过仿真进行验证。

10.1　确定性单臂机械手的 PD+前馈控制

10.1.1　单臂机械手的运动方程

假设连杆的质量均匀分布，质心距连杆的转动中心为 l，连杆运动的粘性摩擦系数为 d，并忽略弹性摩擦，则根据牛顿定律得到其运动方程为

$$I\ddot{\theta} + d\dot{\theta} + mgl\cos\theta = \tau \tag{10.1}$$

式中，mg 为重力，$I = \dfrac{4}{3}ml^2$ 为转动惯量。

10.1.2　控制器的设计

设希望的轨迹为 $\theta_d(t)$，则跟踪误差为：

$$e = \theta - \theta_d \tag{10.2}$$

设误差的动态特性满足特征方程

$$\ddot{e} + a\dot{e} + be = 0 \tag{10.3}$$

式中，a，b 为正的常数。

根据代数稳定性判据，针对二阶系统，当系统闭环特征方程式的系数都大于零时，系统稳定，式（10.3）的跟踪误差 $e(t)$ 收敛于零。

引入辅助控制信号 u，考虑到前馈补偿，令控制律为

$$\tau = u + I\ddot{\theta}_d + d\dot{\theta}_d + mgl\cos\theta \tag{10.4}$$

由式（10.1）和式（10.4）得：

$$u = I\ddot{e} + d\dot{e} \tag{10.5}$$

由式（10.3）和式（10.5），得到 PD 控制：

$$u = (d - aI)\dot{e} - bIe \tag{10.6}$$

将式（10.6）带入式（10.4），得到最终的控制律为：

$$\tau = (d - aI)\dot{e} - bIe + I\ddot{\theta}_d + d\dot{\theta}_d + mgl\cos\theta \tag{10.7}$$

由式（10.7）可见，控制律相当于“PD+前馈控制”。

10.1.3　仿真程序及分析

仿真实例

机械臂的参数为 $m = 1\text{kg}$，$l = 0.25\text{m}$，$d = 2.0\text{N}\cdot\text{m}\cdot\text{s/rad}$，控制器参数选为 $a = 20.0$，

b 25 0。指令信号为 $\sin(2\pi t)$。仿真结果如图 10-1～图 10-4 所示。

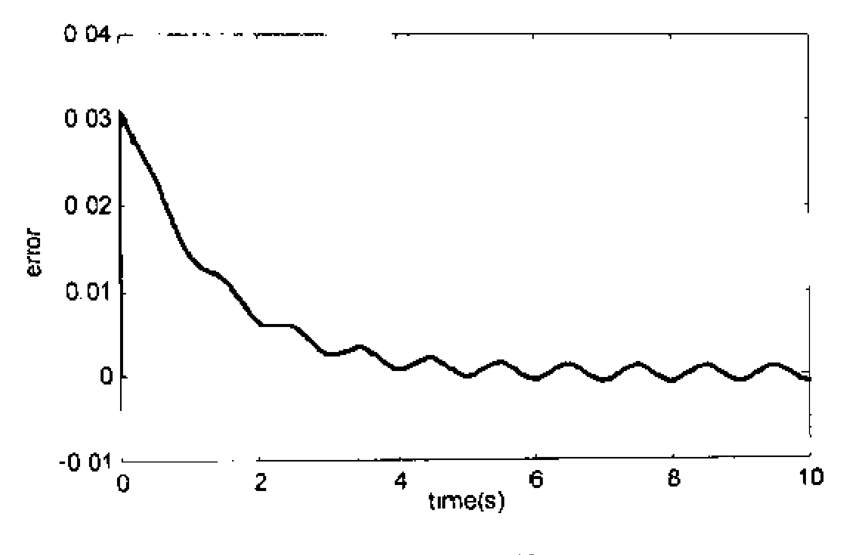


图 10-1 跟踪误差

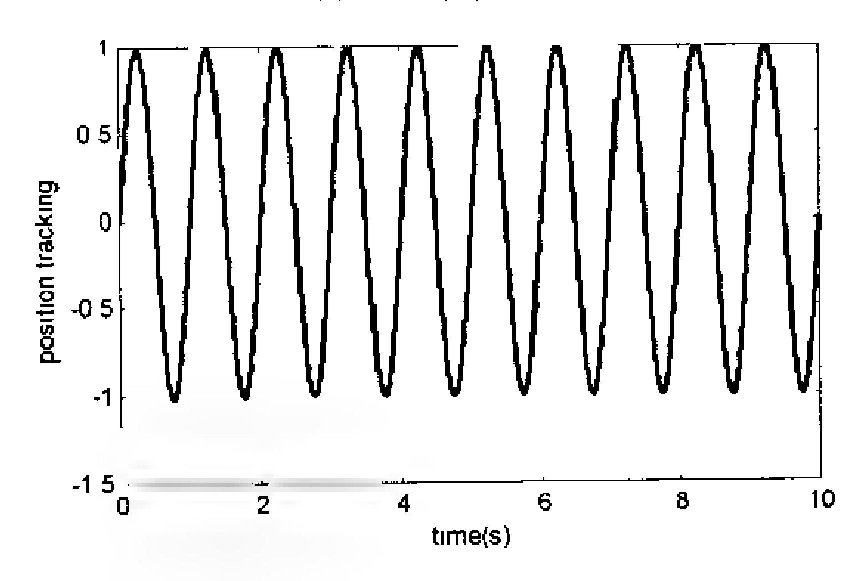


图 10-2 正弦位置跟踪

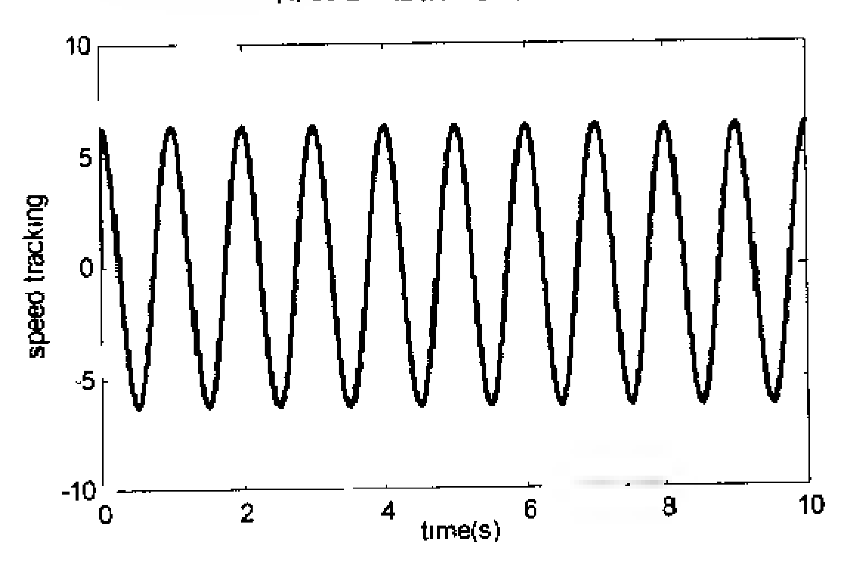


图 10-3 正弦速度跟踪

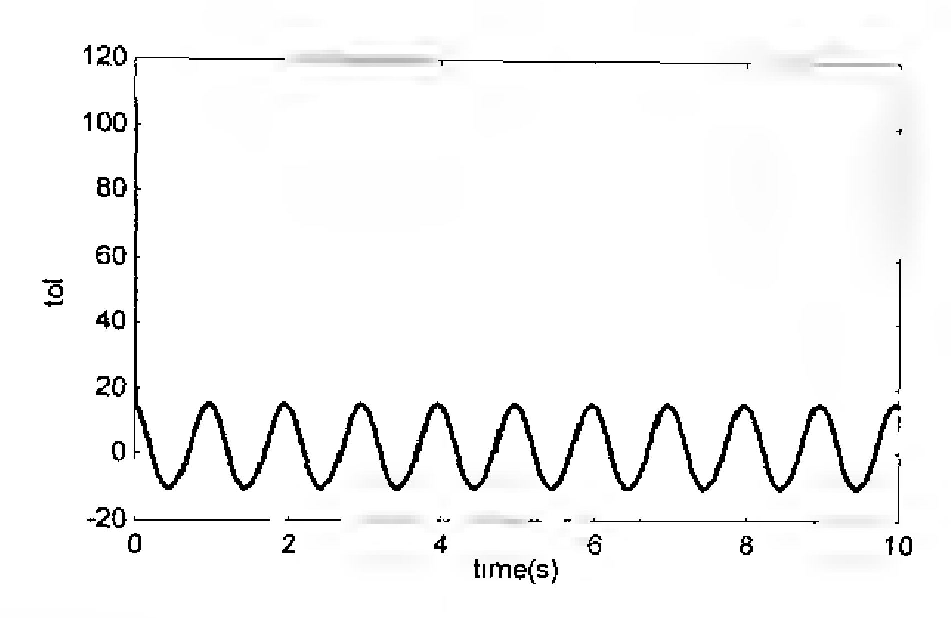


图 10-4 控制器输出

仿真程序由以下四个程序构成。

Simulink 主程序：chap10_1.mdl，如图 10-5 所示

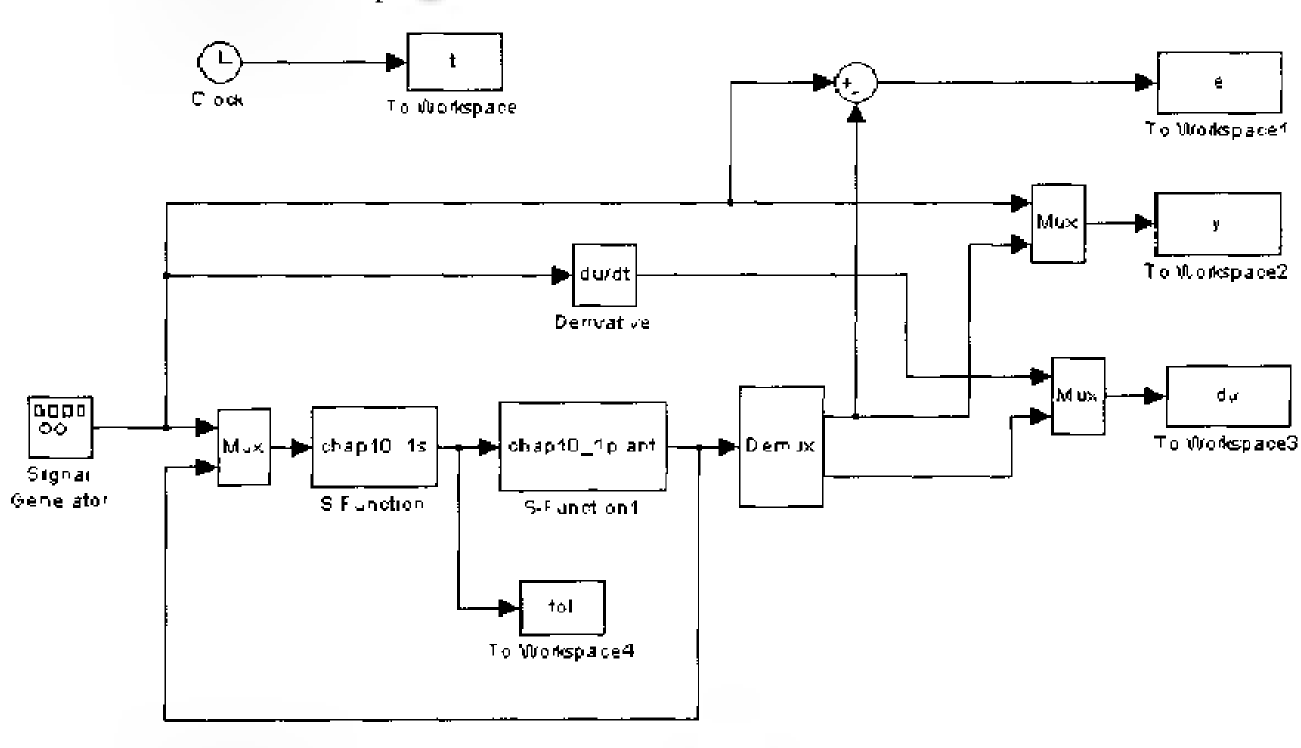


图 10-5 确定条件下机械臂的 Simulink 主程序

S 函数控制程序：chap10_1s.m

```
function [sys,x0,str,ts]=spacemodel(t,x,u,flag)

switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
```

```
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end

function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = [illegible];
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 1;
sizes.NumInputs      = [illegible];
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0; % At least one sample time is needed
sys = simsizes(sizes);
x0 = [];
str = [];
ts = [];

function sys=mdlOutputs(t,x,u)

g=9.8;
m=1;
l=0.25;
d=2.0;
a=200;k=150;
I=4/3*m*l^2;

A=1.0;F=1.0;
r=u(1);
x1=u(2);
x2=u(3);

dr=A*F*2*pi*cos(F*2*pi*t);
ddr=-A*(F*2*pi)^2*sin(F*2*pi*t);

e=x1-r;
de=x2-dr;

tol=-d*a*[illegible]*de-k*I*e+I*ddr+d*dr+m*g*l*cos(x1);
```

```
sys(1)=tol;
```

S 函数被控对象程序：chap10_1plant.m。

```
function [sys,x0,str,ts]=spacemodel(t,x,u,flag)

switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end

function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 2;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 2;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1; % At least one sample time is needed
sys = simsizes(sizes);
x0  = [0;0];
str = [];
ts  = [0 0];

function sys=mdlDerivatives(t,x,u)  %Time varying model

g=9.8;
m=1;
l=0.25;
d=2.0;
I=4/3*m*l^2;

tol=u;
```

```
sys(1)=x(2);
sys(2)=1/I*(-d*x(2)-m*g*l*cos(x(1))+tol(1));

function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);
```

作图子程序：chap10_1plot.m

```
close all;

figure(1);
plot(t,e,'r');
xlabel('time(s)');ylabel('error');

figure(2);
plot(t,y(:,1),'r',t,y(:,2),'k');
xlabel('time(s)');ylabel('position tracking');

figure(3);
plot(t,dy,'r');
xlabel('time(s)');ylabel('speed tracking');

figure(4);
plot(t,tol,'r');
xlabel('time(s)');ylabel('tol');
```

10.2 不确定性单臂机械手的PD+前馈控制

10.2.1 不确定性单臂机械手的运动方程

考虑不确定性单臂机械手，即粘性摩擦系数不准确，且不忽略弹性摩擦时的情况。假设粘性摩擦系数真值为 $\tilde{d}$，且弹性摩擦系数为 δ_0，此时机械臂运动方程为：

$$I\ddot{\theta}+\tilde{d}\dot{\theta}+\delta_0\theta+mgl\cos\theta=\tau \tag{10.8}$$

粘性摩擦系数的不确定性用误差 δ_1 表示，即：

$$\delta_1=\tilde{d}-d \tag{10.9}$$

针对不确定性对象，取 $\delta_1=0.8$，$\delta_0=0.2$，其他参数与10.1节相同，仍采用式(10.7)表示的控制律，仿真结果如图10.6和图10.7所示。可见，由于不确定性的存在，位置跟踪误差和速度跟踪误差均不为零。

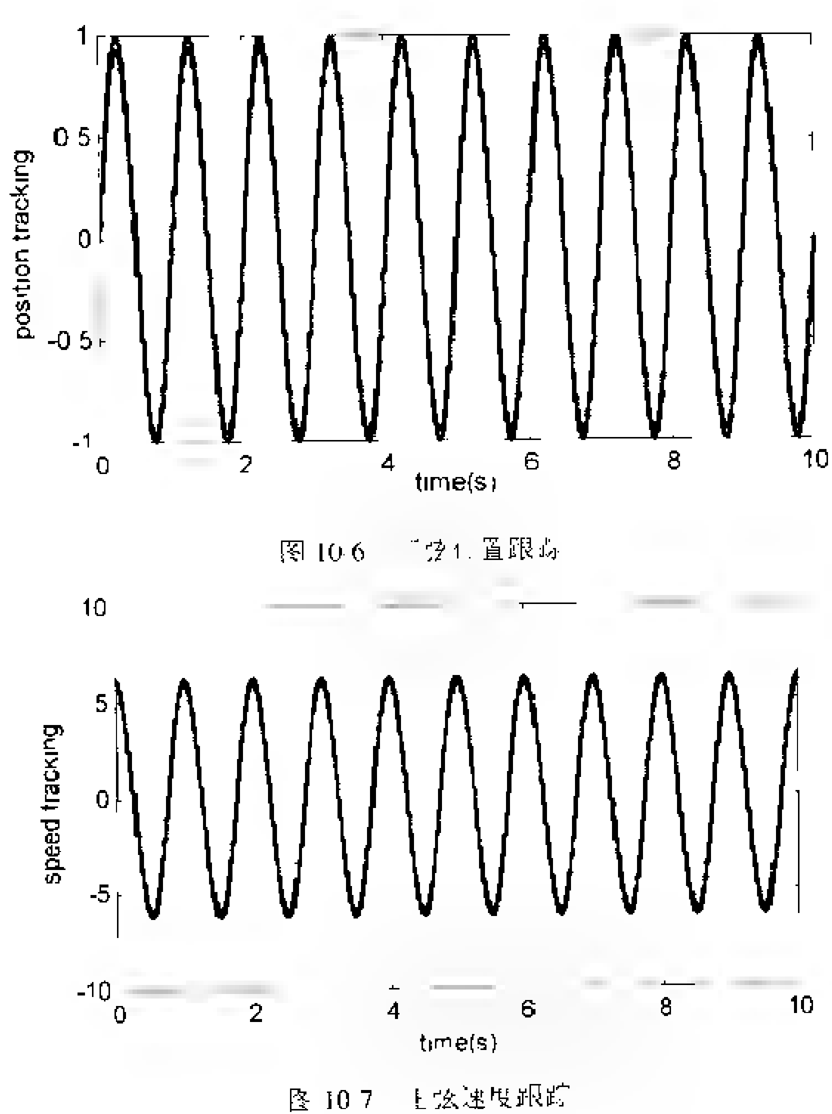


图 10.6 正弦位置跟踪

图 10.7 正弦速度跟踪

10.2.2 仿真程序及分析

Simulink 主程序：chap10_2.mdl，如图 10.8 所示

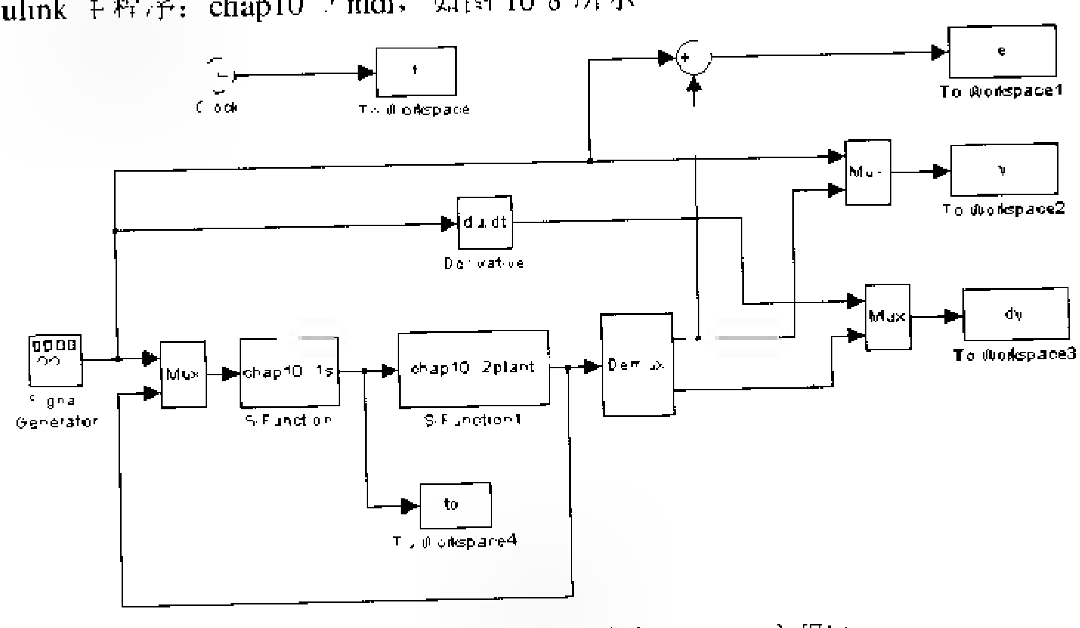


图 10.8 不确定条件下机械臂的 Simulink 主程序

S 函数程序：同 chap10_1s.m。

S 函数被控对象程序：chap10_2plant.m。

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)

switch flag,
case 0
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1
    sys=mdlDerivatives(t,x,u);
case 3
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end

function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 2;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 2;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1; % At least one sample time is needed
sys = simsizes(sizes);
x0  = [0;0];
str = [];
ts  = [0 0];

function sys=mdlDerivatives(t,x,u)   %Time varying model

g=9.8;
m=1;
l=0.25;
d=2.0;
I=4/3*m*l^2;

delta0=[illegible];
d=2.0;
delta1=0.8[illegible];
dp=d+delta1;      %Prediction value

tol=[illegible]
```

```
sys(1)=x(2);
sys(2)=1/I*(-dp*x(2)-delta0*x(1)-m*g*l*cos(x(1))+tol);

function sys=mdlOutputs(t,x,u)
sys(1)=x(1);
sys(2)=x(2);
```

作图程序同 chap10_1plot.m。

10.3 不确定性单臂机械手的 PD 鲁棒控制

10.3.1 控制器设计

针对不确定性单臂机械手的运动方程式（10.8），按“PD+前馈+补偿”的方法设计鲁棒控制器为：

$$\tau = (d - aI)\dot e - bIe + I\ddot\theta_d + d\dot\theta_d + mgl\cos\theta + Iv \tag{10.10}$$

式中，v 为鲁棒补偿项。

将式（10.10）带入式（10.8）得：

$$I(\ddot e + a\dot e + be) - Iv = \delta_1\dot\theta_d - \delta_0\theta_d - \delta_1\dot e - \delta_0 e \tag{10.11}$$

令 $w(\theta_d,\dot\theta_d) = (\delta_1\dot\theta_d + \delta_0\theta_d)/I$，$\Delta f = w(\theta_d,\dot\theta_d) - \frac{\delta_1}{I}\dot e - \frac{\delta_0}{I}e$，则有：

$$\ddot e + a\dot e + be = v + \Delta f \tag{10.12}$$

令 $\boldsymbol{x}^{\mathrm{T}} = [e, \dot e]$，则式（10.12）可表示为：

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}(v + \Delta f) \tag{10.13}$$

式中，$\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ -b & -a \end{bmatrix}$，$\boldsymbol{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$。

在不确定性机械臂运动方程式（10.8）中，假设 δ_0 和 δ_1 为未知，但其上界已知，即满足：

$$|\delta_0| < k_1,\quad |\delta_1| < k_2,\quad |w(\theta_d,\dot\theta_d)| < \rho(\theta_d,\dot\theta_d) \tag{10.14}$$

式中，k_1，k_2 为给定常数，ρ 为给定的界函数。

由于 $|\Delta f| = \left|w(\theta_d,\dot\theta_d) - \frac{\delta_1}{I}\dot e - \frac{\delta_0}{I}e\right| \leqslant |w(\theta_d,\dot\theta_d)| + \left|\frac{\delta_1}{I}\dot e\right| + \left|\frac{\delta_0}{I}e\right| \leqslant \rho(\theta_d,\dot\theta_d) + \frac{1}{I}(k_2|\dot e| + k_1|e|)$

令 $\tilde\rho(\theta_d,\dot\theta_d,\dot e,e) = \rho(\theta_d,\dot\theta_d) + \frac{1}{I}(k_2|\dot e| + k_1|e|)$，则有：

$$|\Delta f| \leqslant \tilde\rho \tag{10.15}$$

10.3.2 稳定性分析

定义 Lyapunov 函数：

$$V(x) = \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{x}$$

由于A为稳定的，则对于任意给定的正定阵Q，存在正定解$P>0$，满足以下Lyapunov方程：

$$A^{\mathrm T}P+PA=-Q \tag{10.16}$$

对Lyapunov函数求导数，得：

$$\dot V=\frac{1}{2}\dot x^{\mathrm T}Px+\frac{1}{2}x^{\mathrm T}P\dot x=\frac{1}{2}(x^{\mathrm T}A^{\mathrm T}+B^{\mathrm T}(v+\Delta f))Px+\frac{1}{2}x^{\mathrm T}P(Ax+B(v+\Delta f))$$

$$=\frac{1}{2}x^{\mathrm T}(A^{\mathrm T}P+PA)x+\frac{1}{2}B^{\mathrm T}(v+\Delta f)Px+\frac{1}{2}x^{\mathrm T}PB(v+\Delta f)$$

$$=\frac{1}{2}x^{\mathrm T}(A^{\mathrm T}P+PA)x+x^{\mathrm T}PB(v+\Delta f)=-\frac{1}{2}x^{\mathrm T}Qx+x^{\mathrm T}PB(v+\Delta f)$$

$$\leqslant -\frac{1}{2}x^{\mathrm T}Qx+x^{\mathrm T}PBv+\left|x^{\mathrm T}PB\right|\Delta f\leqslant -\frac{1}{2}x^{\mathrm T}Qx+x^{\mathrm T}PBv+\left|x^{\mathrm T}PB\right|\tilde\rho$$

令

$$v=-\frac{x^{\mathrm T}PB\,\tilde\rho^{2}}{\left|x^{\mathrm T}PB\right|\tilde\rho+\gamma e^{-\beta t}} \tag{10.17}$$

$$\dot V\leqslant -\frac{1}{2}x^{\mathrm T}Qx-\frac{(x^{\mathrm T}PB)^{2}\tilde\rho^{2}}{\left|x^{\mathrm T}PB\right|\tilde\rho+\gamma e^{-\beta t}}+\left|x^{\mathrm T}PB\right|\tilde\rho=-\frac{1}{2}x^{\mathrm T}Qx+\frac{\left|x^{\mathrm T}PB\right|\tilde\rho}{\left|x^{\mathrm T}PB\right|\tilde\rho+\gamma e^{-\beta t}}\gamma e^{-\beta t}$$

$$\leqslant -\frac{1}{2}x^{\mathrm T}Qx+\gamma e^{-\beta t}$$

式中，γ为一个正的常数，只要取γ值充分小，便可以实现$\dot V<0$。

针对不确定性对象式（10.8），取$\gamma=9$，$\beta=0.10$，$\delta_1=0.2$，$\delta_0=0.2$，其他参数与10.1节相同，采用控制律式（10.10），取$Q=\begin{bmatrix}10 & 0\\ 0 & 1\end{bmatrix}$，则求解Lyapunov方程式（10.16）得$P=\begin{bmatrix}4.875 & 0.2\\ 0.2 & 0.035\end{bmatrix}$。仿真结果如图10-9～图10-13所示。可见，由于采用了鲁棒控制，位置跟踪误差和速度跟踪误差均趋近于零。

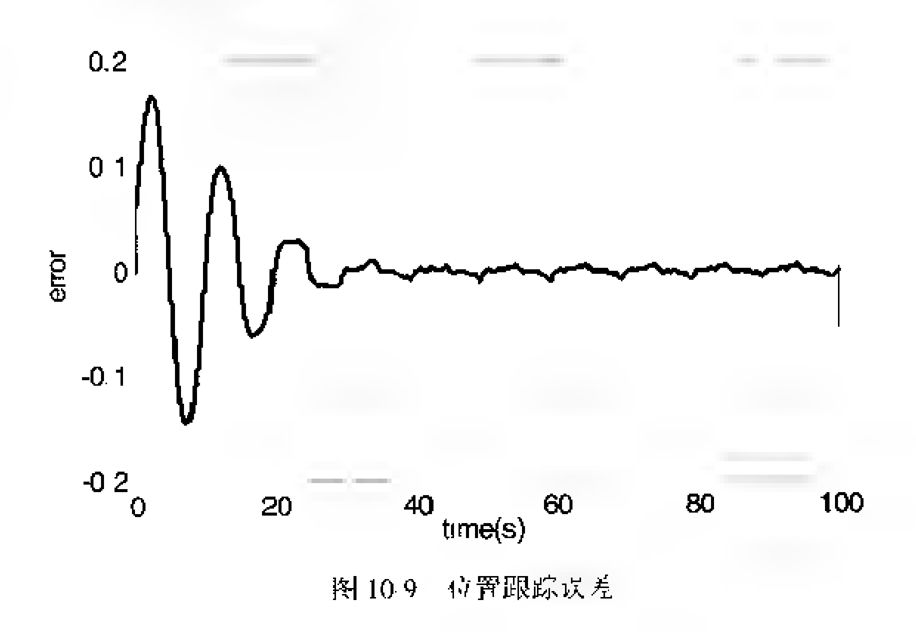


图10-9　位置跟踪误差

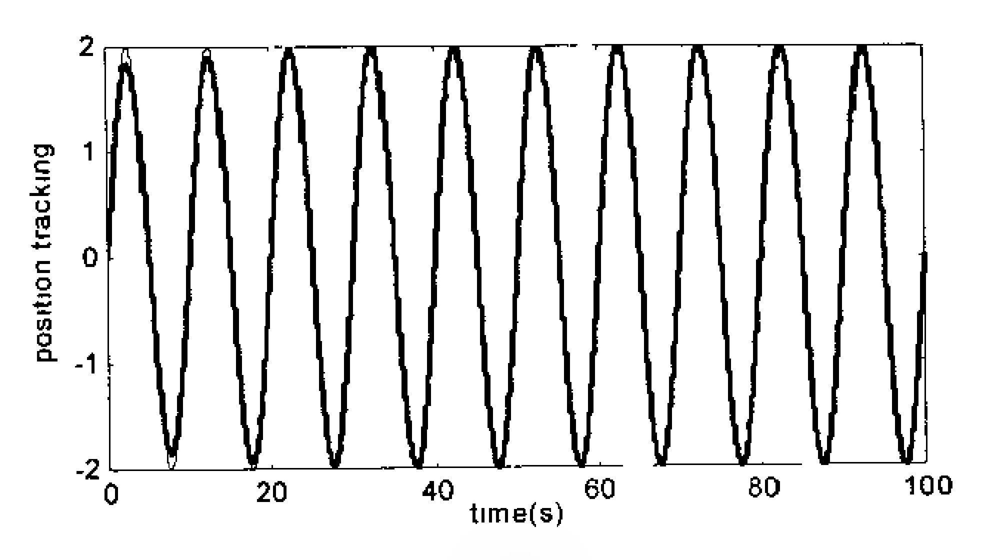


图 10-10　正弦位置跟踪

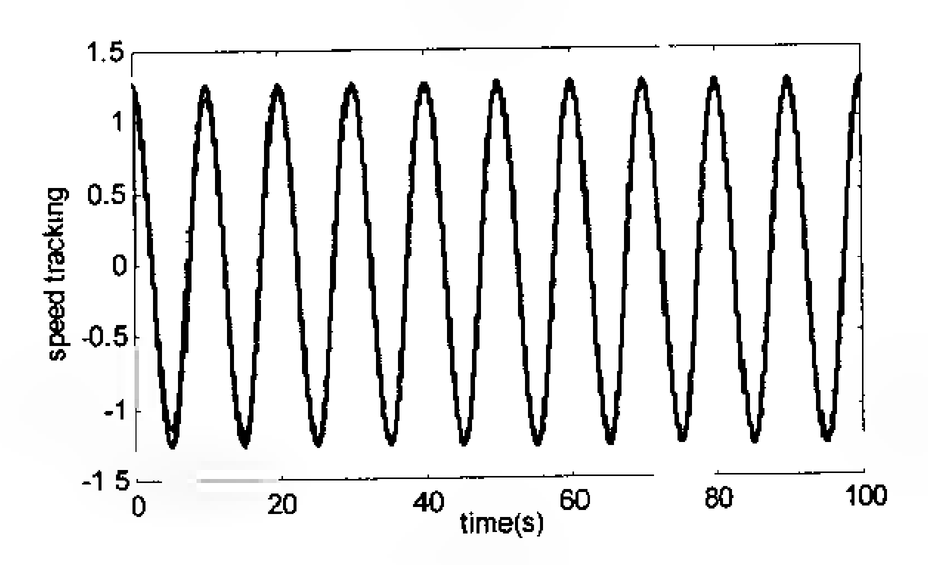


图 10-11　正弦速度跟踪

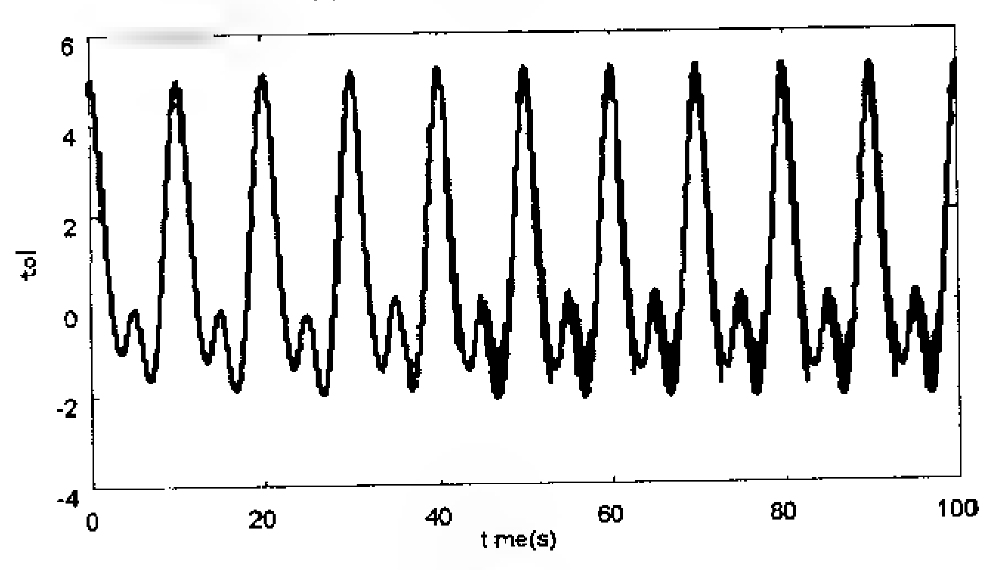


图 10-12　控制器输出

10.3.3 仿真程序及分析

Simulink 主程序：chap10_3.mdl，如图 10-13 所示。

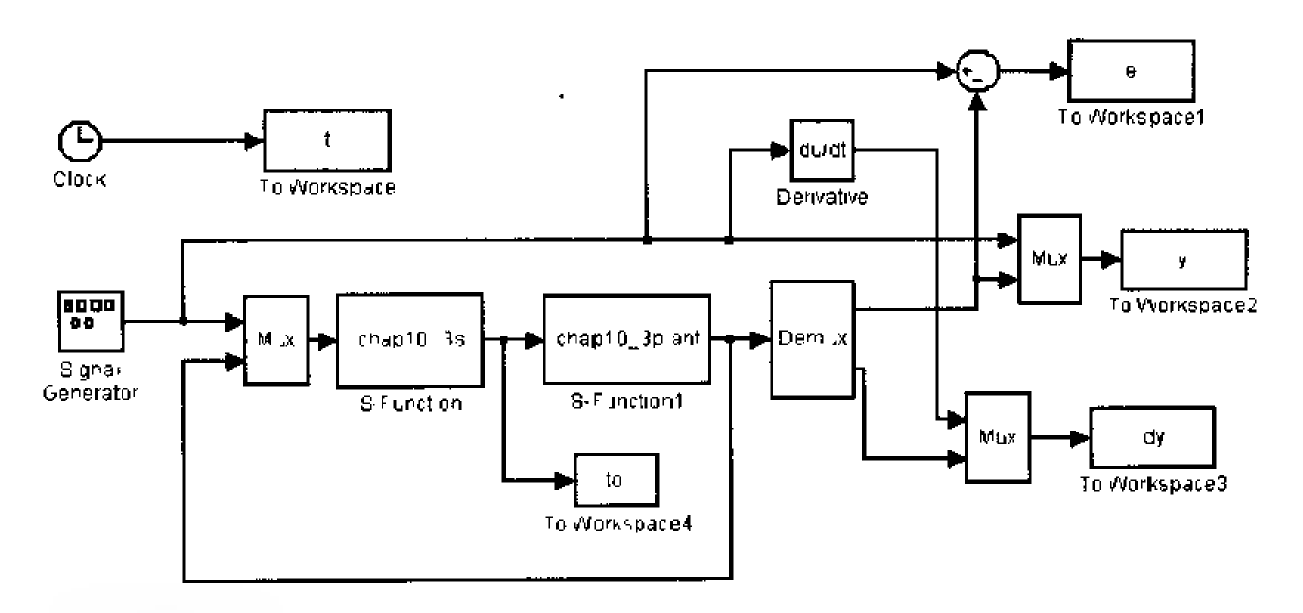


图 10-13 不确定性机械臂的 Simulink 主程序

S 函数控制子程序：chap10_3s.m。

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)

switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end

function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 5;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0 = [];
str = [];
```

```
ts=[0 0];

function sys=mdlOutputs(t,x,u)
g=9.8;
m=1;
l=0.25;
d=2.0;
I=4/3*m*l^2;

AA=2.0;
FF=0.10;
r=u(1);
dr=AA*FF*2*pi*cos(FF*2*pi*t);
ddr=-AA*(FF*2*pi)^2*sin(FF*2*pi*t);

x1=u(2);
x2=u(3);

e=x1-r;
de=x2-dr;

a=20;b=25;
A=[0 1;-b -a];
B=[0;1];

Q=[10 0;
   0 1];
P=lyap(A',Q);
eig(P);

delta0=0.2;
delta1=0.2;

w=-(delta1*dr+delta0*r)/I;
%df=w-delta1/I*de-delta0/I*e;

gama=9;
beta=0.3;

rou=abs(w)+0.10;
rou1=rou+1/I*(abs(de)+abs(e));
```

```
xe=[e;de];
v=-xe'*P*B*rou1^2-abs(xe'*P*B)*rou1+q*um*exp(-beta*t);
%v=0;

d1=delta1+d;
tol=1/d*(a*I*de-b*I*e+I*ddr+d*dr+m*g*l*cos(x1))+I*v;

sys(1)=tol;
```

S函数被控对象子程序：chap10_3plant.m

```
function [sys,x0,str,ts]=spacemodel(t,x,u,flag)

switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end

function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 2;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 2;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = [0;0];
str = [];
ts = [0 0];

function sys=mdlDerivatives(t,x,u)  %Time varying model
g=9.8;
m=1;
l=0.25;
```

```
d 2 ^,
l 4 ^*m*.^2,

delta0 ^.2;
delt.1 ^ .-

d1 del al+a;

^ol .-

.ys  x . ;
^ys 2  'I* tol d *x 2  delta *x(.  m*g*l*c^s x .  .

f.nction s,s mdlO.tputs t,x,.

s/s 1  x 1 ,
sy^ 2  x . .
```

作图程序：chap10_3plot m

```
close all;

figure ^ ;
plot t e,'r );
xl.bel 'time s  ;ylabe.  error  ;

f.g.re 2 ;
plot t , :,1 , r',t,y( ,2  'b  ;
xlabel  t.me(s)  ,ylabel  position tr.ck.ng  ,

fig.re(  ;
plot t,dy :,1), r',t,.y(.,. ,'b' ;
xl.bel('  .e s   ,ylabel  speed tracking' ;

fig.re(4 ;
p ot  ,tol,'r  ,
x abel '^ime s  ;ylabel 'tol  ,
```

10.4　基于 PD 的 N 关节机器人控制

针对 N 关节的机器人控制，陈启军教授等提出了基于 PD 的 3 种常用机器人轨迹跟踪的

控制算法，并证明了算法的稳定性[41]。该控制算法及其稳定性分析具有一定代表性。下面分别介绍这三种控制算法，并通过仿真进行验证。

10.4.1 N 关节机器人运动方程

考虑具有 n 个旋转关节的刚性机器人，其动力学模型为：

$$M(\theta)\ddot{\theta}+C(\theta,\dot{\theta})\dot{\theta}+G(\theta)=\tau \tag{10.18}$$

式中，$\theta \in R^n$ 为关节角位移量，$M(\theta)\in R^n$ 为惯性矩阵，$C(\theta,\dot{\theta})\in R^n$ 表示离心力和哥氏力项，$G(\theta)\in R^n$ 为重力项，$\tau \in R^n$ 为控制力矩。

10.4.2 PD 控制

对式（10.18）所示的系统，考虑如下控制律：

$$\tau=-K_p e - K_v\dot{e} \tag{10.19}$$

其中，

$$e=\theta-\theta_d,\quad e\in R^n$$
$$\dot{e}=\dot{\theta}-\theta_d,\quad e\in R^n$$
$$K_p=\mathrm{diag}(k_{p1}\quad k_{p2}\quad \cdots\quad k_{pn})^T,\quad k_{pi}>0$$
$$K_v=\mathrm{diag}(k_{v1}\quad k_{v2}\quad \cdots\quad k_{vn})^T,\quad k_{vi}>0$$

如果期望跟踪的轨迹速度 $\dot{\theta}_d$ 和加速度 $\ddot{\theta}_d$ 有界，则式（10.19）可保证 e 和 $\dot{e}$ 指数收敛到半径 r_i（$i=1,2$）的封闭球，增大 K_p 和 K_v 可使球半径任意小。

10.4.3 PD 控制+前馈控制

对式（10.18）所示的系统，考虑如下控制律：

$$\tau=-K_p e-K_v e+M(\theta)\ddot{\theta}_d+C(\theta,\dot{\theta})\dot{\theta}_d+G(\theta) \tag{10.20}$$

如果期望跟踪的轨迹速度 θ_d 和加速度 $\ddot{\theta}_d$ 有界，则式（10.20）可保证 e 和 $\dot{e}$ 指数收敛到 0。

10.4.4 PD 控制+修正前馈控制

对式（10.18）所示的系统，考虑如下控制律：

$$\tau=-K_p e-K_v\dot{e}+M(\theta_d)\theta_d+C(\theta_d,\dot{\theta}_d)\dot{\theta}_d+G(\theta_d) \tag{10.21}$$

如果期望跟踪的轨迹速度 θ_d 和加速度 $\dot{\theta}_d$ 有界，则式（10.21）可保证 e 和 e 指数收敛到半径 r_i（$i=1,2$）的封闭球，增大 K_p 和 K_v 可使球半径任意小。

10.4.5 仿真程序及分析

仿真实例

设机器人的动力学模型为：

$$\begin{bmatrix} D_{11} & D_{12}\cos(\theta_1-\theta_2) \\ D_{21}\cos(\theta_1-\theta_2) & D_{22} \end{bmatrix}\begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix}+$$

$$\begin{bmatrix} -\dot{\theta}_2\sin(\theta_2) & -(\dot{\theta}_1+\dot{\theta}_2)\sin(\theta_2) \\ \dot{\theta}_1\sin(\theta_2) & 0 \end{bmatrix}\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}+$$

$$\begin{bmatrix} D_{12}\dot{\theta}_2^2\sin(\theta_1-\theta_2) \\ -D_{12}\dot{\theta}_1^2\sin(\theta_1-\theta_2) \end{bmatrix}=\tau$$

式中，$D_{11}=2.462\text{kg}\cdot\text{m}^2$，$D_{22}=0.362\text{kg}\cdot\text{m}^2$，$D_{12}=D_{21}=0.147\text{kg}\cdot\text{m}^2$

在仿真过程中，指令信号选为 $F=2$，控制参数为：$k_{p1}=3000$，$k_{p2}=2000$，$k_{v1}=230$，$k_{v2}=210$。S=1,2,3 分别对应三种控制律，采用 PD 控制+修正前馈控制式（10.21）进行仿真（S=3），仿真结果如图 10-14～图 10-17 所示。

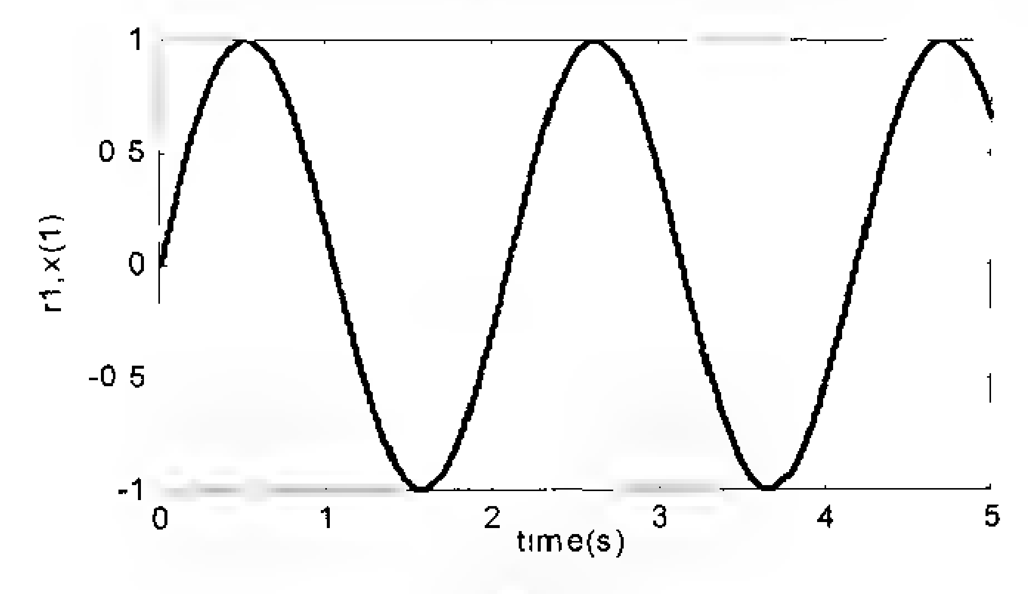


图 10-14　关节 1 的位置跟踪

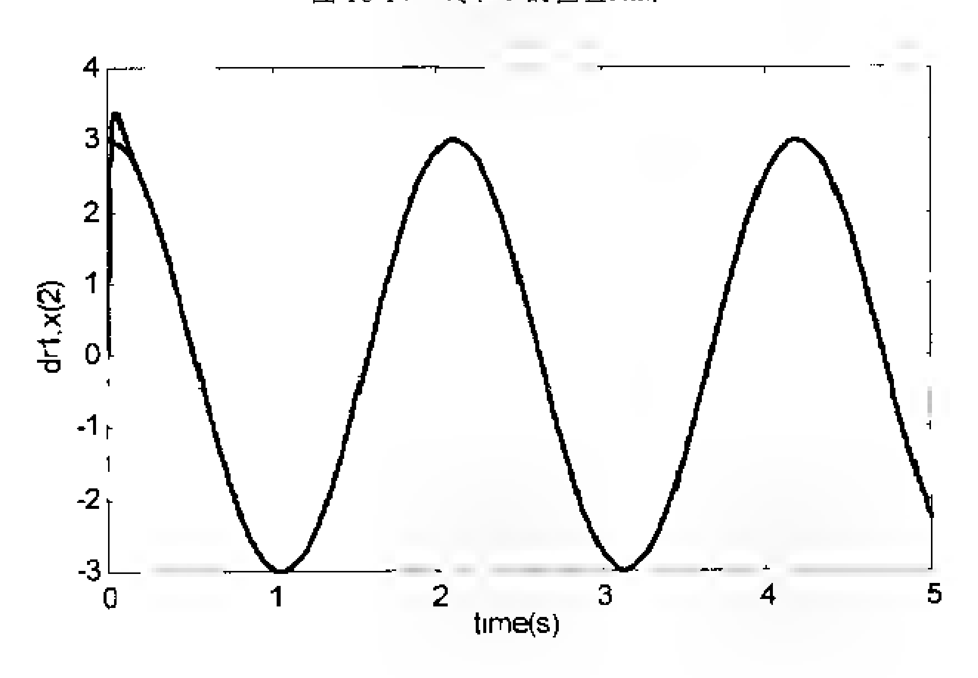


图 10-15　关节 1 的速度跟踪

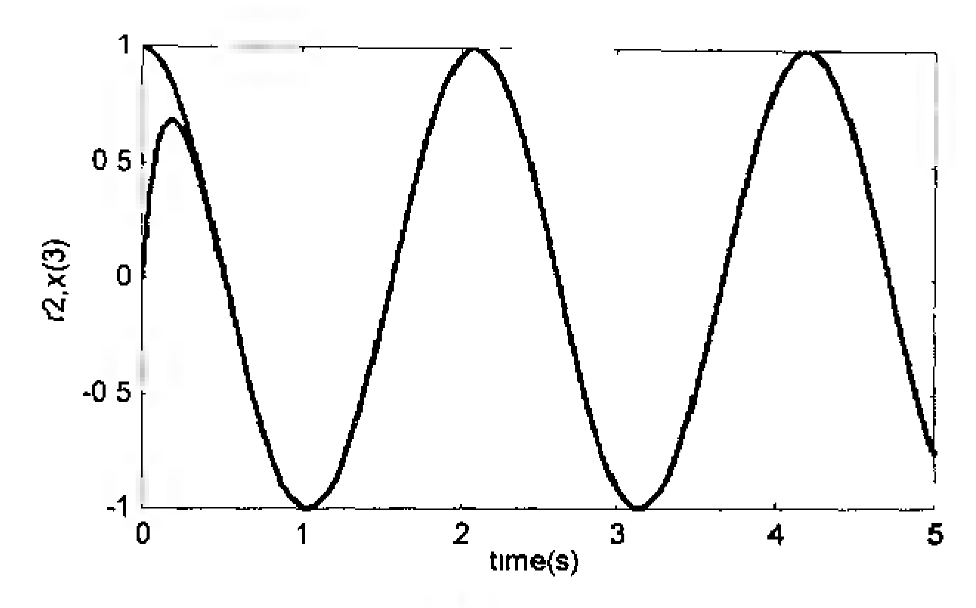


图 10-16　关节 2 的位置跟踪

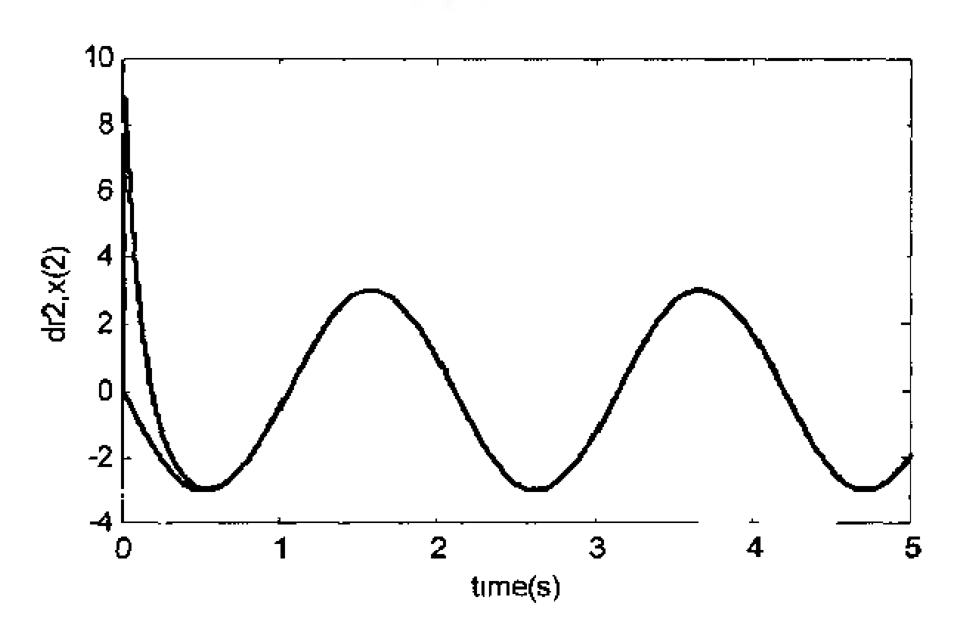


图 10-17　关节 2 的速度跟踪

仿真程序的控制主程序：chap10_4.m。

```
%PD Control for Robotic Manipulator ,2003/04/15
clear all;
close all;
global F

x0=[0,0,0,0];

ts=0.001;
T=5.0;
TimeSet=[0:ts:T];
para=[];
[t,y]=ode45('chap10_4eq',TimeSet,x0,[],para);
```

```
switch F
case 1
  r1=pi/6;r2=pi/6;
  dr1=0;dr2=0;
case 2
  r1=sin(3*t);r2=cos(3*t);
  dr1=3*cos(3*t);dr2=-3*sin(3*t);
case 3
  r1=sign(sin(3*t));r2=sign(sin(3*t));
  dr1=0;dr2=0;
end

figure(1);
plot(t,r1,'r',t,y(:,1),'b');
xlabel('time(s)');ylabel('r1,x(1)');
figure(2);
plot(t,dr1,'r',t,y(:,2),'b');
xlabel('time(s)');ylabel('dr1,x(2)');

figure(3);
plot(t,r2,'r',t,y(:,3),'b');
xlabel('time(s)');ylabel('r2,x(3)');
figure(4);
plot(t,dr2,'r',t,y(:,4),'b');
xlabel('time(s)');ylabel('dr2,x(4)');
```

控制子程序：chap10_4eq.m。

```
function dx=PlantModel(t,x,flag,para)
global F
dx=zeros(4,1);

F=2;
switch F
case 1
  r1=pi/6;r2=pi/6;
  dr1=0;dr2=0;
  ddr1=0;ddr2=0;
case 2
  r1=sin(3*t);r2=cos(3*t);
  dr1=3*cos(3*t);dr2=-3*sin(3*t);
  ddr1=-3*3*sin(3*t);ddr2=-3*3*cos(3*t);
```

```
case 3
   r1=sign(sin(3*t));r2=sign(sin(3*t));
   dr1=0;dr2=0;
   ddr1=0;ddr2=0;
end
dr=[dr1;dr2];
ddr=[ddr1;ddr2];

e1=x(1)-r1;
de1=x(2)-dr1;
e2=x(3)-r2;
de2=x(4)-dr2;

kp1=3000;kv1=250;
kp2=2000;kv2=210;

D11=2.464;
D22=0.362;
D12=0.147;
D21=0.147;

M=[D11 D12*cos(x(1)-x(3));
    D21*cos(x(1)-x(3)) D22];
G=[D12*x(4)^2*sin(x(1)-x(3));
   D12*x(2)^2*sin(x(1)-x(3))];
C=[-x(4)*sin(x(3)) -(x(2)+x(4))*sin(x(3));
    x(2)*sin(x(3)) 0];

S=3;
switch S
case 1
   u1=-kp1*e1-kv1*de1;
   u2=-kp2*e2-kv2*de2;
case 2
   u1=-kp1*e1-kv1*de1+M(1,:)*ddr+C(1,:)*dr+G(1);
   u2=-kp2*e2-kv2*de2+M(2,:)*ddr+C(2,:)*dr+G(2);
case 3
    Md=[D11 D12*cos(r1-r2);
          D21*cos(r1-r2) D22];
    Gd=[D12*dr1^2*sin(r1-r2);
       -D12*dr2^2*sin(r1-r2)];
   Cd=[-dr2*sin(r2) -(dr1+dr2)*sin(r2);
```

```
    dr1*sin(t)];
  u1=kp1*e1+kv1*de1+Md(1,:)*ddr+C(1,:)*dr+Gd(1);
  u2=kp2*e2+kv2*de2+Md(2,:)*ddr+C(2,:)*dr+Gd(2);
end
tol=[u1;u2];

S=inv(M)*(tol-C*[x(2);x(4)]-G);

dx(1)=x(2);
dx(2)=S(1);
dx(3)=x(4);
dx(4)=S(2);
```

10.5 机器人的鲁棒自适应 PD 控制

具有强耦合性和非线性的机器人系统而言，线性 PD 控制是最为简单且行之有效的控制方法，在工业机器人中得到了广泛的应用。但实践表明，线性 PD 控制往往使驱动机构有很大的初始输出，而驱动机构（通常是电机）不可能提供过大的初始力矩，且机械臂本身所承受的最大力矩也是有限的，这将使通过增大 PD 控制系数来进一步提高系统的性能受到限制。鉴于此，很多非线性 PD 控制方法被提出，但常规的非线性 PD 控制器只有单纯的 PD 项，要求比例和微分项的系数仍较大，存在输出力矩较大的问题。

焦晓红等提出了一种自适应鲁棒 PD 控制策略，避免了初始输出力矩过大的弊端[42]。该控制器由非线性 PD 反馈和补偿控制两部分构成，机器人不确定动力学部分由回归矩阵构成的自适应控制器进行补偿，并针对机器人有界扰动的上确界是否已知设计了两种不同的扰动补偿法。该控制策略的优点在于当初始误差较大时，PD 反馈起主要作用，通过非线性 PD 控制，避免了过大初始力矩输出；当误差较小时，自适应控制器起着主要的作用，从而保证系统具有良好的动态性能。

10.5.1 机器人动力学模型及其结构特性

考虑一个 N 关节的机器人，其动态性能可以由以下二阶非线性微分方程描述：

$$M(\theta)\ddot{\theta}+C(\theta,\dot{\theta})\dot{\theta}+G(\theta)+D(\dot{\theta})+\omega=\tau \tag{10.22}$$

式中，$\theta\in R^n$ 为关节角位移量，$M(\theta)\in R^{n\times n}$ 为机器人的惯性矩阵，$C(\theta,\dot{\theta})\in R^n$ 表示离心力和哥氏力，$G(\theta)\in R^n$ 为重力项，$D(\dot{\theta})\in R^n$ 表示粘性摩擦力和干摩擦力，$\tau\in R^n$ 为控制力矩，$\omega\in R^n$ 为各种误差和扰动。

机器人系统的动力学特性如下：

特性 1：$\dot{M}(\theta)-2C(\theta,\dot{\theta})$ 是一个斜对称矩阵。

特性 2：惯性矩阵 $M(\theta)$ 是对称正定矩阵，存在正数 m_1，m_2 满足如下不等式：

$$m_1\|x\|^2\le x^{\mathrm{T}}M(\theta)x\le m_2\|x\|^2 \tag{10.23}$$

特性 3：存在一个依赖于机械手参数的参数向量，使得 $M(\theta)$，$C(\theta,\dot{\theta})$，$G(\theta)$，$D(\dot{\theta})$ 满

定线性关系：

$$M(\theta)\vartheta + C(\theta,\dot{\theta})\rho + G(\theta) + D(\dot{\theta}) = \Phi(\theta,\dot{\theta},\rho,\vartheta)P \tag{10.24}$$

式中，$\Phi(\theta,\dot{\theta},\vartheta,\rho)\in R^{n\times m}$ 为已知关节变量函数的回归矩阵，它是机器人广义坐标及其各阶导数的已知函数矩阵；$P\in R^{n}$ 是描述机器人质量特性的未知定常参数向量。

假设 1：$\theta_d \in R^n$ 为期望的关节角位移，θ_d 的一阶导数和二阶导数存在。

假设 2：误差和扰动 ω 的范数满足：

$$\|\omega\| \le d_1 + d_2\|e\| + d_3\|\dot{e}\| \tag{10.25}$$

式中，d_1，d_2，d_3 分别为正常数，$e=\theta-\theta_d$，$\dot{e}=\dot{\theta}-\dot{\theta}_d$ 分别为跟踪误差和跟踪误差导数。

10.5.2 控制器的设计

分别引入变量 y 和 θ_r，并令：

$$y = \dot{e} + \gamma e \tag{10.26}$$

$$\dot{\theta}_r = \dot{\theta}_d - \gamma e \tag{10.27}$$

式中，常数 $\gamma>0$，则可推出：

$$y = \dot{\theta} - \dot{\theta}_r \tag{10.28}$$

将式（10.24）、式（10.26）～式（10.28）代入式（10.22）中，可得：

$$M(\theta)\dot{y} + C(\theta,\dot{\theta})y = \tau - \Phi(\theta,\dot{\theta},\dot{\theta}_r,\ddot{\theta}_r)P - \omega \tag{10.29}$$

1．扰动信号的上确界已知时控制器的设计

对于式（10.22）所示的机器人系统，在误差扰动信号的上确界已知时，采用控制器式（10.30）和式（10.31），可保证系统全局渐进稳定。

$$\tau = -K_p e - K_v \dot{e} + \Phi(\theta,\dot{\theta},\dot{\theta}_r,\ddot{\theta}_r)\hat{P} + u \tag{10.30}$$

$$u = [u_1 \cdots u_n]^T,\quad u_i = -(d_1 + d_2\|e\| + d_3\|\dot{e}\|)\mathrm{sgn}(y_i) \tag{10.31}$$

$\hat{P}$ 的参数估计律取：

$$\dot{\hat{P}} = -\Gamma\Phi^T(\theta,\dot{\theta},\dot{\theta}_r,\ddot{\theta}_r)y \tag{10.32}$$

其中，

$$K_p = K_{p1} + K_{p2}B_p(e),\quad K_v = K_{v1} + K_{v2}B_v(\dot{e})$$

$$K_{p1} = \mathrm{diag}(k_{p11},k_{p12},\cdots,k_{p1n}),\quad K_{p2} = \mathrm{diag}(k_{p21},k_{p22},\cdots,k_{p2n})$$

$$K_{v1} = \mathrm{diag}(k_{v11},k_{v12},\cdots,k_{v1n}),\quad K_{v2} = \mathrm{diag}(k_{v21},k_{v22},\cdots,k_{v2n})$$

$$B_p(e) = \mathrm{diag}\left(\frac{1}{\alpha_1+|e_1|},\frac{1}{\alpha_2+|e_2|},\cdots,\frac{1}{\alpha_n+|e_n|}\right),\quad B_v(\dot{e}) = \mathrm{diag}\left(\frac{1}{\beta_1+|\dot{e}_1|},\frac{1}{\beta_2+|\dot{e}_2|},\cdots,\frac{1}{\beta_n+|\dot{e}_n|}\right)$$

式中，k_{p1i}，k_{p2i}，k_{v1i}，k_{v2i}，α_i，$\beta_i (i=1,2,\cdots,n)$ 均大于零，Γ 为正定对称阵。

在参考文献[42]研究成果的基础上，这里给出稳定性证明的详细推导过程。

稳定性证明：

令 $\tilde{P} = \hat{P} - P$，定义 Lyapunov 函数为：

$$V = \frac{1}{2}\left(y^T M(\theta)y + e^T(K_{p1} + \gamma K_{v1})e + \tilde{P}^T\Gamma^{-1}\tilde{P}\right)$$

考虑 M 为对称正定矩阵，则有：

$$\left(y^{T}M(\theta)y\right)' = \dot{y}^{T}M(\theta)y + y^{T}\dot{M}(\theta)y + y^{T}M(\theta)\dot{y} = 2y^{T}M(\theta)\dot{y} + y^{T}\dot{M}(\theta)y$$

同理，可得：

$$\left(e^{T}(K_{p1}+\gamma K_{v1})e\right)' = 2e^{T}(K_{p1}+\gamma K_{v1})\dot{e}$$

$$\left(\tilde{P}^{T}\Gamma^{-1}\tilde{P}\right)' = 2\tilde{P}^{T}\Gamma^{-1}\dot{\tilde{P}}$$

从而有：

$$\dot{V} = y^{T}M(\theta)\dot{y} + \frac{1}{2}y^{T}\dot{M}(\theta)y + e^{T}(K_{p1}+\gamma K_{v1})\dot{e} + \tilde{P}^{T}\Gamma^{-1}\dot{\tilde{P}}$$

由式（10.29）、式（10.30）得：

$$\begin{aligned} y^{T}M(\theta)\dot{y} &= y^{T}\left[\tau - \Phi(\theta,\dot{\theta},\dot{\theta}_r,\ddot{\theta}_r)P - \omega - C(\theta,\dot{\theta})y\right] \\ &= y^{T}\left[-K_p e - K_v\dot{e} + \Phi(\theta,\dot{\theta},\dot{\theta}_r,\ddot{\theta}_r)\hat{P} + u - \Phi(\theta,\dot{\theta},\dot{\theta}_r,\ddot{\theta}_r)P - \omega - C(\theta,\dot{\theta})y\right] \\ &= y^{T}\left[-K_p e - K_v\dot{e} + \Phi(\theta,\dot{\theta},\dot{\theta}_r,\ddot{\theta}_r)\tilde{P} + u - \omega\right] - y^{T}C(\theta,\dot{\theta})y \end{aligned}$$

由 K_p 和 K_v 的定义得：

$$K_{p1}+\gamma K_{v1} = K_p - K_{p2}B_p(e) + \gamma(K_v - K_{v2}B_v(\dot{e}))$$

$$e^{T}(K_{p1}+\gamma K_{v1})\dot{e} = e^{T}K_p\dot{e} - e^{T}\left(K_{p2}B_p(e) + \gamma K_{v2}B_v(\dot{e})\right)\dot{e} + e^{T}\gamma K_v\dot{e}$$

则有：

$$\dot{V} = y^{T}\left[-K_p e - K_v\dot{e} + \Phi(\theta,\dot{\theta},\dot{\theta}_r,\ddot{\theta}_r)\tilde{P} + u - \omega\right] - y^{T}C(\theta,\dot{\theta})y + \frac{1}{2}y^{T}\dot{M}(\theta)y +$$

$$e^{T}K_p\dot{e} - e^{T}\left(K_{p2}B_p(e) + \gamma K_{v2}B_v(\dot{e})\right)\dot{e} + e^{T}\gamma K_v\dot{e} + \tilde{P}^{T}\Gamma^{-1}\dot{\tilde{P}}$$

由机器人特性1可知：

$$-y^{T}C(\theta,\dot{\theta})y + \frac{1}{2}y^{T}\dot{M}(\theta)y = \frac{1}{2}y^{T}\left(\dot{M}(\theta) - 2C(\theta,\dot{\theta})\right)y = 0$$

则

$$\dot{V} = y^{T}(-K_p e - K_v\dot{e}) + e^{T}\gamma K_v\dot{e} + e^{T}K_p\dot{e} - e^{T}\left(K_{p2}B_p(e) + \gamma K_{v2}B_v(\dot{e})\right)\dot{e} +$$

$$y^{T}\left[\Phi(\theta,\dot{\theta},\dot{\theta}_r,\ddot{\theta}_r)\tilde{P} + u - \omega\right] + \tilde{P}^{T}\Gamma^{-1}\dot{\tilde{P}}$$

由上知得：

$$y^{T} = \dot{e}^{T} + \gamma e^{T}$$

则有：

$$y^{T}(-K_p e - K_v\dot{e}) = -\dot{e}^{T}K_p e - \dot{e}^{T}K_v\dot{e} - \gamma e^{T}K_p e - \gamma e^{T}K_v\dot{e}$$

将上式带入 $\dot{V}$ 中，得：

$$\dot{V} = -\dot{e}^{T}K_v\dot{e} - \gamma e^{T}K_p e - e^{T}\left(K_{p2}B_p(e) + \gamma K_{v2}B_v(\dot{e})\right)\dot{e} + y^{T}(u - \omega) + y^{T}\Phi(\theta,\dot{\theta},\dot{\theta}_r,\ddot{\theta}_r)\tilde{P} + \tilde{P}^{T}\Gamma^{-1}\dot{\tilde{P}}$$

考虑到 $y^{T}\Phi(\theta,\dot{\theta},\dot{\theta}_r,\ddot{\theta}_r)\tilde{P} = \tilde{P}^{T}\Phi^{T}(\theta,\dot{\theta},\dot{\theta}_r,\ddot{\theta}_r)y$、$\dot{\tilde{P}} = \dot{\hat{P}}$ 及式（10.32），得：

$$y^{T}\Phi(\theta,\dot{\theta},\dot{\theta}_r,\ddot{\theta}_r)\tilde{P} + \tilde{P}^{T}\Gamma^{-1}\dot{\tilde{P}} = \tilde{P}^{T}\Phi^{T}(\theta,\dot{\theta},\dot{\theta}_r,\ddot{\theta}_r)y + \tilde{P}^{T}\Gamma^{-1}\dot{\tilde{P}} = \tilde{P}^{T}\left(\Phi^{T}(\theta,\dot{\theta},\dot{\theta}_r,\ddot{\theta}_r)y + \Gamma^{-1}\dot{\hat{P}}\right) = 0$$

则有：

$$\dot{V} = -\gamma e^{T}K_p e - \dot{e}^{T}K_v\dot{e} - e^{T}\left(K_{p2}B_p(e) + \gamma K_{v2}B_v(\dot{e})\right)\dot{e} + y^{T}(u - \omega)$$

其中，

$$\gamma e^{\mathrm{T}} K_{\mathrm{p}} e = \gamma \begin{bmatrix} e_1 & e_2 & \cdots & e_n \end{bmatrix} \left(K_{\mathrm{p}} + K_{\mathrm{p2}} B_{\mathrm{p}}(e)\right) \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix}$$

$$K_{\mathrm{p2}} B_{\mathrm{p}}(e) = \begin{bmatrix} k_{\mathrm{p21}} & & \\ & \ddots & \\ & & k_{\mathrm{p2}n} \end{bmatrix} \begin{bmatrix} \dfrac{1}{\alpha_1 + |e_1|} & & \\ & \ddots & \\ & & \dfrac{1}{\alpha_n + |e_n|} \end{bmatrix} = \begin{bmatrix} \dfrac{k_{\mathrm{p21}}}{\alpha_1 + |e_1|} & & \\ & \ddots & \\ & & \dfrac{k_{\mathrm{p2}n}}{\alpha_n + |e_n|} \end{bmatrix}$$

$$K_{\mathrm{p1}} + K_{\mathrm{p2}} B_{\mathrm{p}}(e) = \begin{bmatrix} k_{\mathrm{p11}} + \dfrac{k_{\mathrm{p21}}}{\alpha_1 + |e_1|} & & \\ & \ddots & \\ & & k_{\mathrm{p1}n} + \dfrac{k_{\mathrm{p2}n}}{\alpha_n + |e_n|} \end{bmatrix}$$

$$\gamma e^{\mathrm{T}} K_{\mathrm{p}} e = \gamma \begin{bmatrix} e_1 & e_2 & \cdots & e_n \end{bmatrix} \left(K_{\mathrm{p}} + K_{\mathrm{p2}} B_{\mathrm{p}}(e)\right) \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} = \gamma \sum_{i}^{n} \left(K_{\mathrm{p1}i} + \frac{k_{\mathrm{p2}i}}{\alpha_i + |e_i|} \right) e_i^2$$

同理:

$$\dot{e}^{\mathrm{T}} K_{\mathrm{v}} \dot{e} = \sum_{i=1}^{n} \left(k_{\mathrm{v1}i} + \frac{k_{\mathrm{v2}i}}{\beta_i + |\dot{e}_i|} \right) \dot{e}_i^2$$

$$e^{\mathrm{T}} \left(K_{\mathrm{p2}} B_{\mathrm{p}}(e) + \gamma K_{\mathrm{v2}} B_{\mathrm{v}}(\dot{e})\right) \dot{e} = \sum_{i=1}^{n} \left(\frac{k_{\mathrm{p2}i}}{\alpha_i + |e_i|} + \frac{\gamma k_{\mathrm{v2}i}}{\beta_i + |\dot{e}_i|} \right) e_i \dot{e}_i$$

则有:

$$\dot{V} = -\sum_{i=1}^{n} \left(\gamma \left(k_{\mathrm{p1}i} + \frac{k_{\mathrm{p2}i}}{\alpha_i + |e_i|} \right) e_i^2 \right) - \sum_{i=1}^{n} \left(\left(k_{\mathrm{v1}i} + \frac{k_{\mathrm{v2}i}}{\beta_i + |\dot{e}_i|} \right) \dot{e}_i^2 \right) - \sum_{i=1}^{n} \left(\left(\frac{k_{\mathrm{p2}i}}{\alpha_i + |e_i|} + \frac{\gamma k_{\mathrm{v2}i}}{\beta_i + |\dot{e}_i|} \right) e_i \dot{e}_i \right) + y^{\mathrm{T}} (u - \omega)$$

由于 $e_i \dot{e}_i \le \frac{1}{2}\left(e_i^2 + \dot{e}_i^2\right)$,则有:

$$\sum_{i=1}^{n} \left(\left(\frac{k_{\mathrm{p2}i}}{\alpha_i + |e_i|} + \frac{\gamma k_{\mathrm{v2}i}}{\beta_i + |\dot{e}_i|} \right) e_i \dot{e}_i \right) \le \frac{1}{2} \sum_{i=1}^{n} \left(\left(\frac{k_{\mathrm{p2}i}}{\alpha_i + |e_i|} + \frac{\gamma k_{\mathrm{v2}i}}{\beta_i + |\dot{e}_i|} \right) e_i^2 \right) + \frac{1}{2} \sum_{i=1}^{n} \left(\left(\frac{k_{\mathrm{p2}i}}{\alpha_i + |e_i|} + \frac{\gamma k_{\mathrm{v2}i}}{\beta_i + |\dot{e}_i|} \right) \dot{e}_i^2 \right)$$

$$\dot{V} \le -\sum_{i=1}^{n} \left(\gamma \left(k_{\mathrm{p1}i} + \frac{k_{\mathrm{p2}i}}{\alpha_i + |e_i|} \right) - \frac{1}{2} \left(\frac{k_{\mathrm{p2}i}}{\alpha_i + |e_i|} + \frac{\gamma k_{\mathrm{v2}i}}{\beta_i + |\dot{e}_i|} \right) \right) e_i^2$$

$$- \sum_{i=1}^{n} \left(\left(k_{\mathrm{v1}i} + \frac{k_{\mathrm{v2}i}}{\beta_i + |\dot{e}_i|} \right) - \frac{1}{2} \left(\frac{k_{\mathrm{p2}i}}{\alpha_i + |e_i|} + \frac{\gamma k_{\mathrm{v2}i}}{\beta_i + |\dot{e}_i|} \right) \right) \dot{e}_i^2 + y^{\mathrm{T}} (u - \omega)$$

当 $2 \ge \gamma \ge \frac{1}{2}$ 时,有:

$$\gamma \frac{k_{\mathrm{p2}i}}{\alpha_i + |e_i|} - \frac{1}{2} \frac{k_{\mathrm{p2}i}}{\alpha_i + |e_i|} \ge 0$$

$$\frac{k_{v2i}}{\beta_i+|\dot e_i|}-\frac{1}{2}\frac{\gamma k_{v2i}}{\beta_i+|\dot e_i|}\geqslant 0$$

此时，

$$\dot V < -\sum_{i=1}^{n}\left(\gamma\left(k_{p1i}-\frac{1}{2}\frac{k_{v2i}}{\beta_i+|e_i|}\right)e_i^2+\left(k_{v1i}-\frac{1}{2}\frac{k_{p2i}}{\alpha_i+|e_i|}\right)\dot e_i^2\right)+y^T(u-\omega)$$

$$< -\sum_{i=1}^{n}\left(\gamma\left(k_{p1i}-\frac{1}{2}\frac{k_{v2i}}{\beta_i}\right)e_i^2+\left(k_{v1i}-\frac{1}{2}\frac{k_{p2i}}{\alpha_i}\right)\dot e_i^2\right)+y^T(u-\omega)$$

由于

$$y^Tu=\sum_{i=1}^{n}y_i\left[-(d_1+d_2\|e\|+d_3\|\dot e\|)\mathrm{sgn}(y_i)\right]=-\sum_{i=1}^{n}\left[(d_1+d_2\|e\|+d_3\|\dot e\|)|y_i|\right]<\sum_{i=1}^{n}-\|w\|\cdot|y_i|$$

$$y^Tw<\|y^T\|\cdot\|w\|$$

其中，$\|y^T\|=\sum_{i=1}^{n}|y_i|$。

则有：

$$y^T(u-w)<-\sum_{i=1}^{n}\|w\||y_i|+\|y^T\|\|w\|=0$$

$$\dot V<-\sum_{i=1}^{n}\left(\gamma\left(k_{p1i}-\frac{1}{2}\frac{k_{v2i}}{\beta_i}\right)e_i^2+\left(k_{v1i}-\frac{1}{2}\frac{k_{p2i}}{\alpha_i}\right)\dot e_i^2\right)$$

只要 k_{p1i}，k_{v1i}，α_i，β_i 的取值满足：

$$k_{p1i}-\frac{k_{v2i}}{2\beta_i}>0 \tag{10.33}$$

$$k_{v1i}-\frac{k_{p2i}}{2\alpha_i}>0 \tag{10.34}$$

就可以得到：

$$\dot V<0$$

则由所定义的 Lyapunov 函数容易得到 $\lim_{t\to\infty}e=0$，$\lim_{t\to\infty}\dot e=0$，控制器全局渐进稳定。

2. 扰动信号的上确界未知时控制器的设计

当误差扰动信号ω的上确界为未知时，设计控制器为：

$$\tau=-K_p e-K_v\dot e+\Phi(\theta,\dot\theta,\dot\theta_r,\ddot\theta_r)\alpha+u \tag{10.35}$$

$$u=-\frac{(\hat d f)^2}{\hat d f\|y\|+\varepsilon^2}y \tag{10.36}$$

$$\dot{\hat d}=\gamma_1 f\|y\|,\quad \hat d(0)=0 \tag{10.37}$$

$$\dot{\varepsilon}=-\gamma_2\varepsilon,\quad \varepsilon(0)=0 \tag{10.38}$$

式中，K_p，K_v 的取值同式（10.30），并保证满足式（10.33）和式（10.34），P 的估计值通过式（10.32）求得，$d=d_1+d_2+d_3$，$\tilde{d}=d-\hat{d}$，$f=\max(1,\|e\|,\|\dot{e}\|)$，$\hat{d}$ 为 d 的估计值，γ_1，γ_2 均为任意的正常数。

对式（10.22）表示的机器人系统，并且误差扰动信号的上确界未知时，采用式（10.35）～式（10.38）表示的控制律可保证系统全局渐进稳定。

证明：

定义 Lyapunov 函数：

$$V=\frac{1}{2}\left(y^{\mathrm{T}}M(\theta)y+e^{\mathrm{T}}\left(K_{\mathrm{p}1}+\gamma K_{\mathrm{v}1}\right)e+\tilde{P}^{\mathrm{T}}\Gamma^{-1}\tilde{P}\right)+\frac{1}{2}\left(\gamma_1^{-1}\tilde{d}^2+\gamma_2^{-1}\varepsilon^2\right)$$

由扰动信号上确界已知时控制器分析的推导可得：

$$\dot{V}\leqslant y^{\mathrm{T}}(u-\omega)+\gamma_1^{-1}\tilde{d}\dot{\tilde{d}}+\gamma_2^{-1}\varepsilon\dot{\varepsilon}$$

将控制律式（10.35）～式（10.38）带入上式得：

$$\dot{V}\leqslant y^{\mathrm{T}}u-y^{\mathrm{T}}\omega+\gamma_1^{-1}\tilde{d}\dot{\tilde{d}}+\gamma_2^{-1}\varepsilon\dot{\varepsilon}=y^{\mathrm{T}}\left(-\frac{\left(\hat{d}f\right)^2}{\hat{d}f\|y\|+\varepsilon^2}\right)y-y^{\mathrm{T}}\omega+\gamma_1^{-1}\tilde{d}\dot{\tilde{d}}-\varepsilon^2$$

由于

$$y^{\mathrm{T}}y=\|y\|^2$$

$$y^{\mathrm{T}}\omega\leqslant\|y\|\cdot\|\omega\|$$

$$\|\omega\|\leqslant d_1+d_2\|e\|+d_3\|\dot{e}\|\leqslant d\cdot f$$

$$\dot{\tilde{d}}=-\dot{\hat{d}}=-\gamma_1 f\|y\|$$

$$\begin{aligned}\dot{V}&\leqslant-\frac{\left(\hat{d}f\right)^2\|y\|^2}{\hat{d}f\|y\|+\varepsilon^2}+\|y\|\|\omega\|-\gamma_1^{-1}\tilde{d}\dot{\hat{d}}-\varepsilon^2\leqslant-\frac{\left(\hat{d}f\right)^2\|y\|^2}{\hat{d}f\|y\|+\varepsilon^2}+\|y\|df-\tilde{d}f\|y\|-\varepsilon^2\\&=-\frac{\left(\hat{d}f\right)^2\|y\|^2}{\hat{d}f\|y\|+\varepsilon^2}+f\|y\|\hat{d}-\varepsilon^2=\frac{-\left(\hat{d}f\right)^2\|y\|^2+\left(\hat{d}f\right)^2\|y\|^2+\varepsilon^2 f\|y\|\hat{d}-\varepsilon^2\hat{d}f\|y\|-\varepsilon^4}{\hat{d}f\|y\|+\varepsilon^2}\\&=-\frac{\varepsilon^4}{\hat{d}f\|y\|+\varepsilon^2}\end{aligned}$$

由 $\hat{d}$ 的定义可知：$\hat{d}>0$，则有：

$$\dot{V}<0$$

10.5.3 仿真程序及分析

仿真实例

选二关节机器人系统（不考虑摩擦力），其动力学模型为：

$$M(\theta)\ddot{\theta}+C(\theta,\dot{\theta})\dot{\theta}+G(\theta)+\omega=\tau$$

$$M(\theta)=\begin{bmatrix}v+q_1+2\gamma\cos(\theta_2) & q_1+q_2\cos(\theta_2)\\ q_1+q_2\cos(\theta_2) & q_1\end{bmatrix}$$

$$C(\theta,\dot{\theta})=\begin{bmatrix}-q_2\dot{\theta}_2\sin(\theta_2) & -q_2(\dot{\theta}_1+\dot{\theta}_2)\sin(\theta_2)\\ q_2\dot{\theta}_1\sin(\theta_2) & 0\end{bmatrix}$$

$$G(\theta)=\begin{bmatrix}15g\cos\theta_1+8.75g\cos(\theta_1+\theta_2)\\ 8.75g\cos(\theta_1+\theta_2)\end{bmatrix}$$

式中，$v=13.33$，$q_1=8.98$，$q_2=8.75$，$g=9.8$。[illegible]，位置指令和系统的初始状态分别为：

$$d_1=2,\ d_2=3,\ d_3=6$$

$$\omega=d_1+d_2\|e\|+d_3\|\dot{e}\|$$

$$\begin{cases}\theta_{1d}=\cos(\pi t)\\ \theta_{2d}=\sin(\pi t)\end{cases}$$

$$\begin{bmatrix}\theta_1\\ \theta_2\\ \dot{\theta}_1\\ \dot{\theta}_2\end{bmatrix}=\begin{bmatrix}0.6\\ 0.3\\ 0.5\\ 0.5\end{bmatrix}$$

控制参数取：

$$K_{p1}=\mathrm{diag}(180,190),\quad K_{p2}=\mathrm{diag}(150,150)$$

$$K_{v1}=\mathrm{diag}(180,180),\quad K_{v2}=\mathrm{diag}(150,150)$$

$$\alpha_i=\beta_i=1\quad(i=1,2),\quad \gamma=1.5$$

采用 S 函数进行控制器和被控对象的设计，按扰动上确界已知和未知两种情况进行仿真。

仿真方法一

设扰动的上确界为已知，[illegible]式（10.30）和式（10.31）进行仿真，仿真结果如图 10-18～图 10-21 所示。

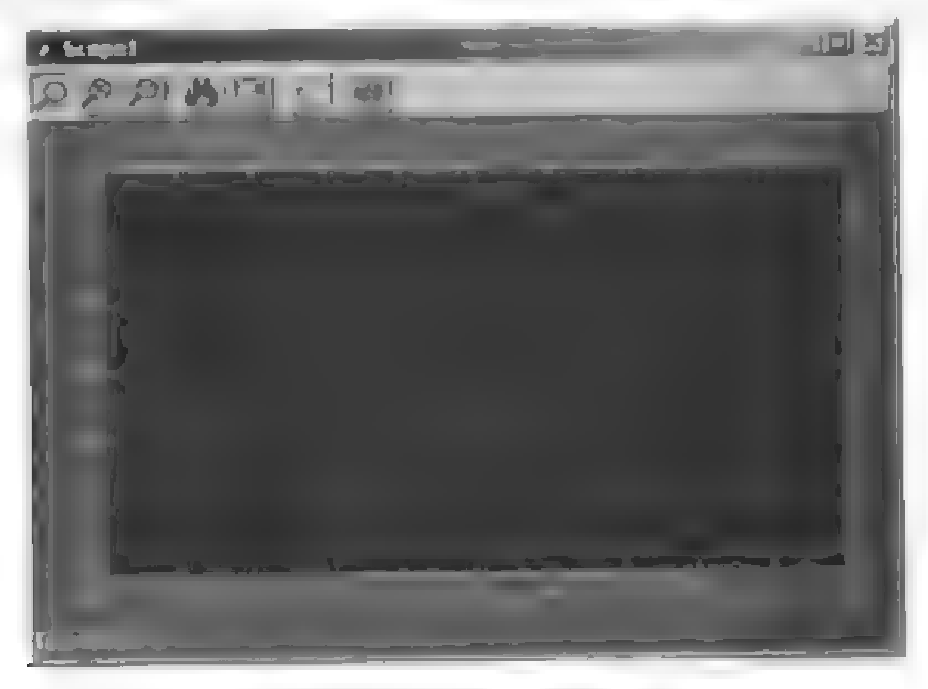


图 10-18 [illegible]

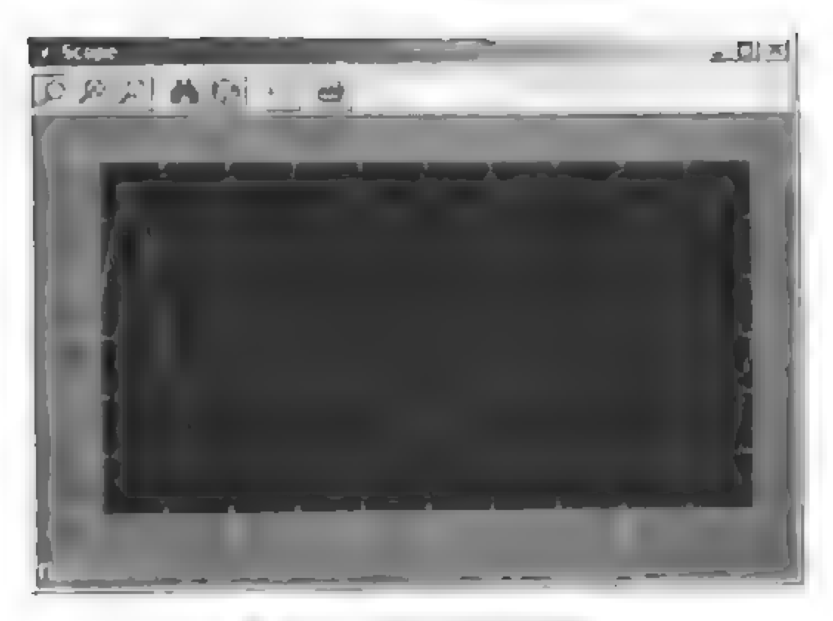


图 10-19 [illegible]

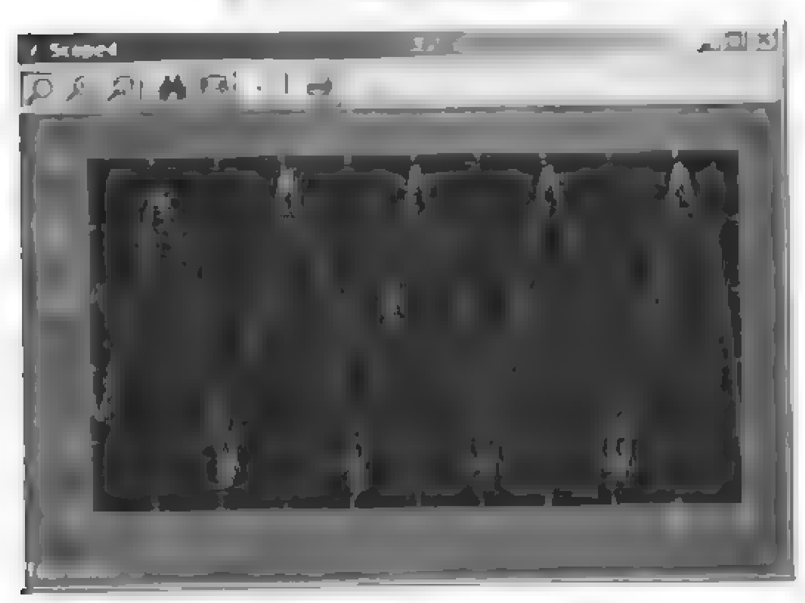


图 10-20 关节1的控制输入

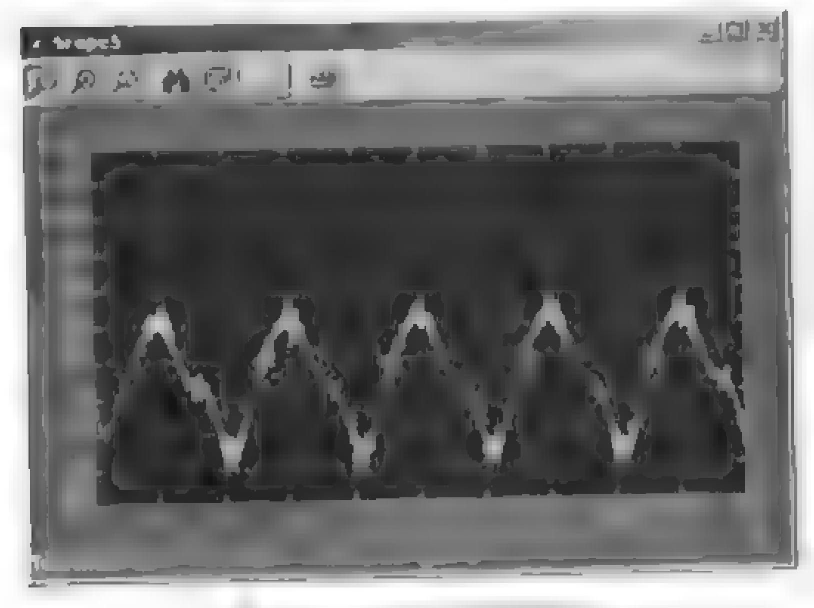


图 10-21 [illegible]

仿真程序的控制主程序：chap10_5.mdl，如图 10-22 所示。

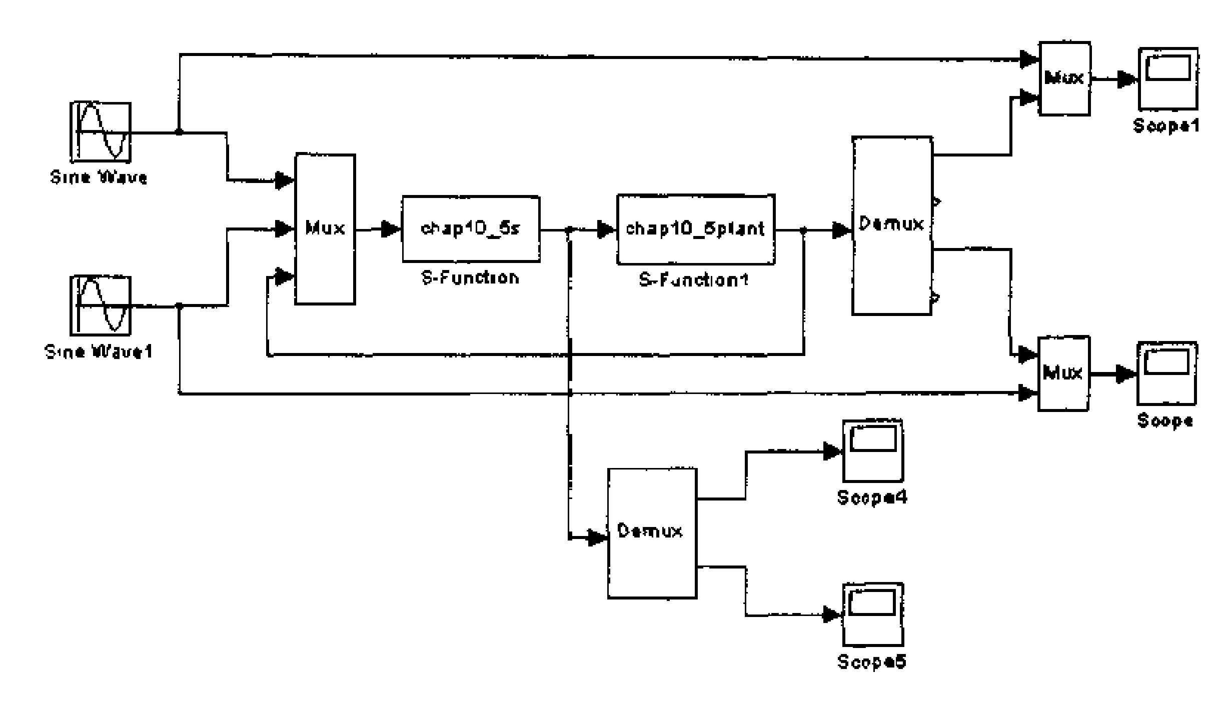


图 10-22 Simulink 主程序

S 函数子程序：chap10_5s.m。

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)

switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end

function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumOutputs     = 2;
sizes.NumInputs      = 6;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [0 0];
```

```
function sys=mdlOutputs(t,x,u)
R1=u(1);
dr1=-pi*sin(pi*t);
ddr1=-pi^2*cos(pi*t);
R2=u(2);
dr2=pi*cos(pi*t);
ddr2=-pi^2*sin(pi*t);

x(1)=u(3);
x(2)=u(4);
x(3)=u(5);
x(4)=u(6);

e1=x(1)-R1;
e2=x(3)-R2;
de1=x(2)-dr1;
de2=x(4)-dr2;

v=13.33;
q1=8.98;
q2=8.75;
g=9.8;

M=[v+q1+2*q2*cos(x(3)) q1+q2*cos(x(3));
   q1+q2*cos(x(3)) q1];
C=[-q2*x(4)*sin(x(3)) -q2*(x(2)+x(4))*sin(x(3));
    q2*x(2)*sin(x(3)) 0];
G=[15*g*cos(x(1))+8.75*g*cos(x(1)+x(3));
   8.75*g*cos(x(1)+x(3))];
d1=2;d2=3;d3=6;

gama=1.5;
y1=de1+gama*e1;
y2=de2+gama*e2;

u1=-(d1+d2*norm([e1,e2])+d3*norm([de1,de2]))*sign(y1);
u2=-(d1+d2*norm([e1,e2])+d3*norm([de1,de2]))*sign(y2);

dr=[dr1;dr2]-gama*[e1;e2];
ddr=[ddr1;ddr2]-gama*[de1;de2];
```

```
Kp1=[180,0;0,190];
Kp2=[150,0;0,150];
Kv1=[180,0;0,180];
Kv2=[150,0;0,150];

alfa1=1;alfa2=1;
beta1=1;beta2=1;

fy=M*ddr+C*dr+G;
tol(1)=-(Kp1(1,1)+Kp2(1,1)/(alfa1+abs(e1)))*e1-(Kv1(1,1)+Kv2(1,1)/(beta1+
abs(de1)))*de1+fy(1)+u1;
tol(2)=-(Kp1(2,2)+Kp2(2,2)/(alfa2+abs(e2)))*e2-(Kv1(2,2)+Kv2(2,2)/(beta2+
abs(de2)))*de2+fy(2)+u2;

sys(1)=tol(1);
sys(2)=tol(2);
```

被控对象子程序：chap10_5plant.m。

```
%S-function for continuous state equation
function [sys,x0,str,ts]=s_function(t,x,u,flag)

switch flag,
%Initialization
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
%Outputs
  case 3,
    sys=mdlOutputs(t,x,u);
%Unhandled flags
  case {2, 4, 9 }
    sys = [];
%Unexpected flags
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end

%mdlInitializeSizes
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 4;
sizes.NumDiscStates  = 0;
```

```
sizes.NumOutputs     = 4;
sizes.NumInputs      = 2;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 0;

sys=simsizes(sizes);
x0=[0.6;0.3;0.5;0.5];
str=[];
ts=[];

function sys=mdlDerivatives(t,x,u)
R1=cos(pi*t);
dr1=-pi*sin(pi*t);
ddr1=-pi^2*cos(pi*t);
R2=sin(pi*t);
dr2=pi*cos(pi*t);
ddr2=-pi^2*sin(pi*t);

e1=x(1)-R1;
e2=x(3)-R2;
de1=x(2)-dr1;
de2=x(4)-dr2;

v=13.33;
q1=8.98;
q2=8.75;
g=9.8;

M=[v+q1+2*q2*cos(x(3)) q1+q2*cos(x(3));
   q1+q2*cos(x(3)) q1];
C=[-q2*x(4)*sin(x(3))  -q2*(x(2)+x(4))*sin(x(3));
    q2*x(2)*sin(x(3))  0];
G=[15*g*cos(x(1))+8.75*g*cos(x(1)+x(3));
   8.75*g*cos(x(1)+x(3))];
d1=2;d2=3;d3=6;
w=[d1+d2*norm([e1,e2])+d3*norm([de1,de2])];

tol(1)=u(1);
tol(2)=u(2);

S=inv(M)*(tol'-C*[x(2);x(4)]-G-w);

sys(1)=x(2);
```

仿真方法二

设扰动的上确界为未知，用控制律（式（10.35）～式（10.38））进行仿真，仿真结果如图 10-23～图 10-27 所示。

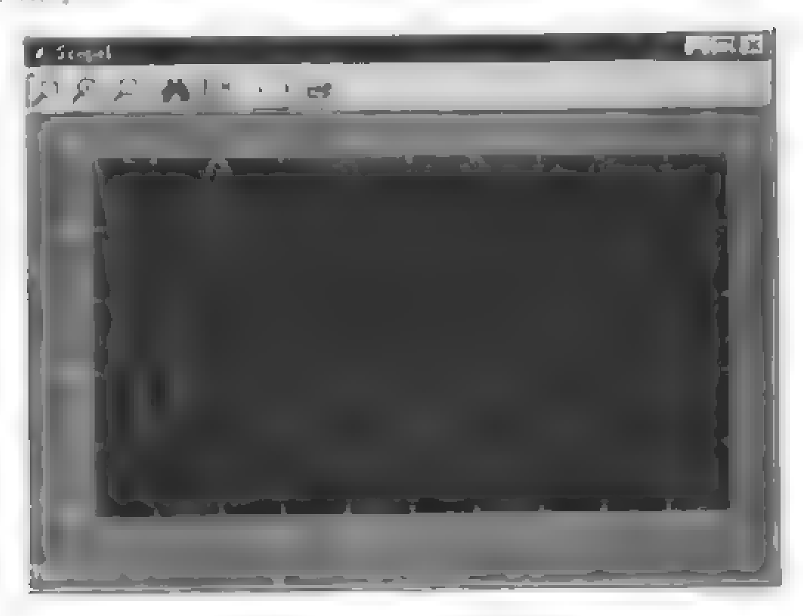


图 10-23 [illegible]

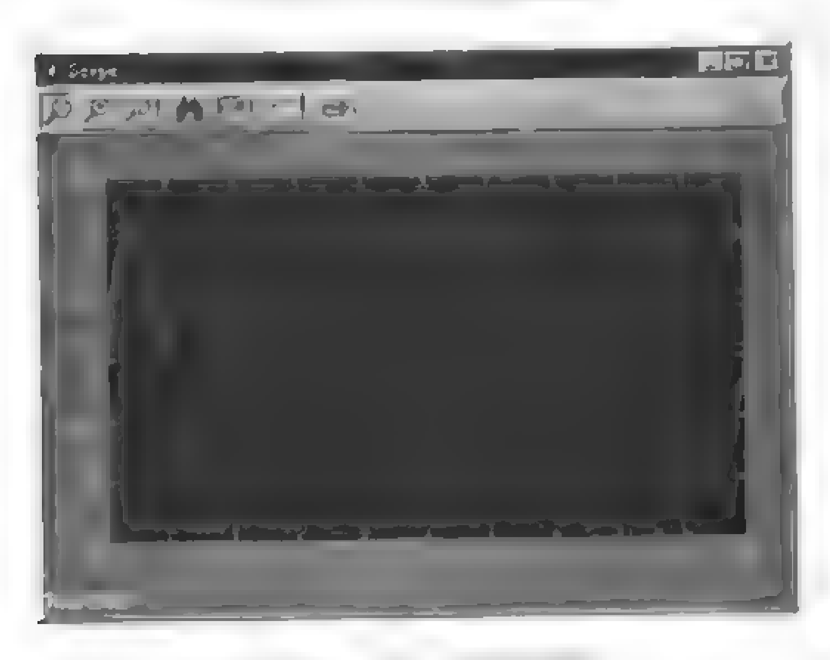


图 10-24 [illegible]

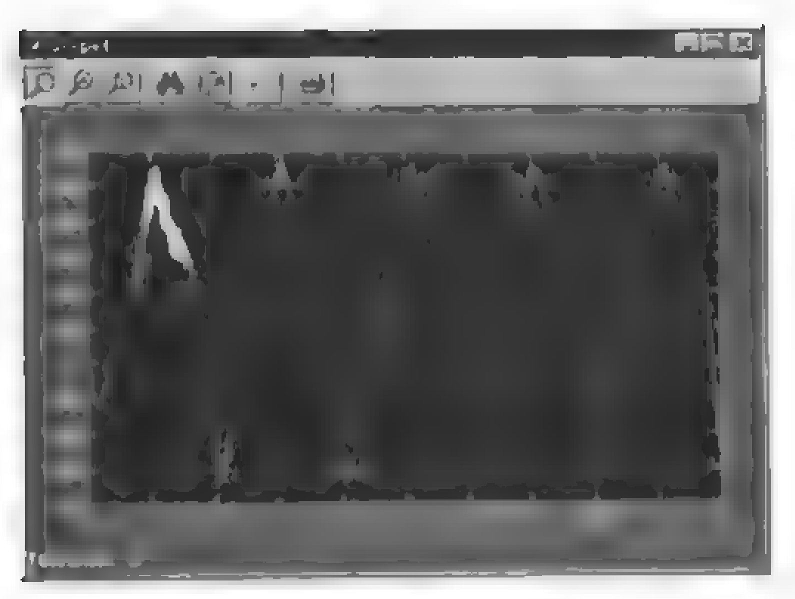


图 10-25　关节 1 的控制输入



图 10-26　关节 2 的控制输入

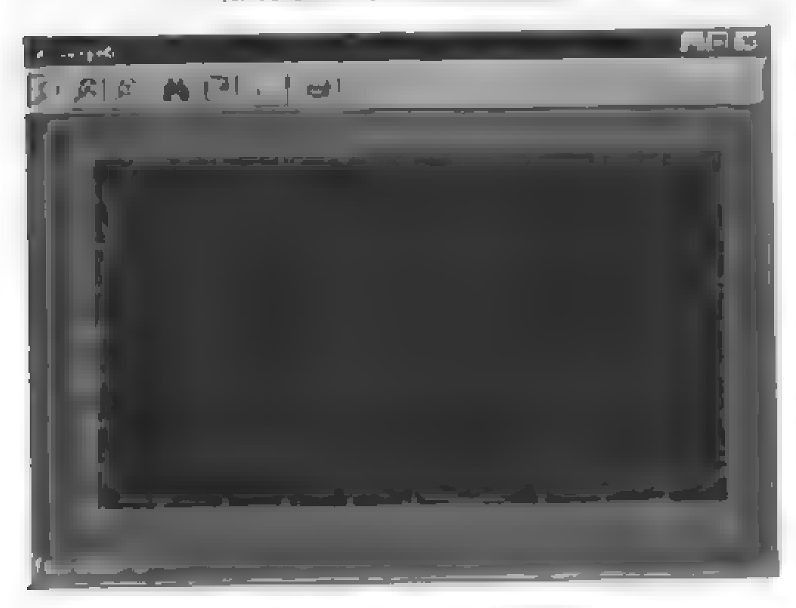


图 10-27　[illegible]

仿真程序的控制主程序：chap10_6.mdl，如图 10-28 所示。

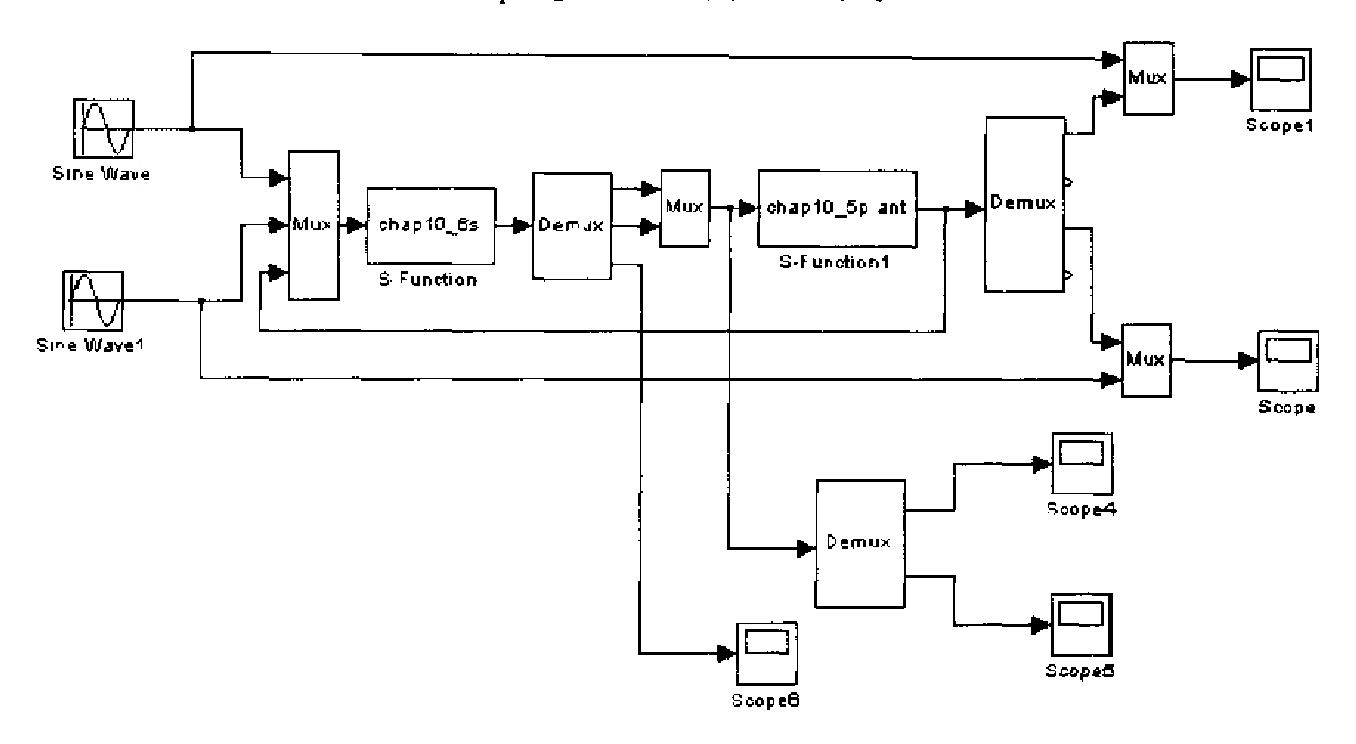


图 10-28 Simulink 主程序

S 函数子程序：chap10_6s.m。

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)

switch flag,
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
case 1,
    sys=mdlDerivatives(t,x,u);
case 3,
    sys=mdlOutputs(t,x,u);
case {2,4,9}
    sys=[];
otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end

function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 2;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 3;
sizes.NumInputs      = 6;
sizes.DirFeedthrough = 0;
```

```
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = [0;0];
str = [];
ts = [0 0];

function sys=mdlDerivatives(t,x,u)
R1=u(1);
dr1=-pi*sin(pi*t);
ddr1=-pi^2*cos(pi*t);
R2=u(2);
dr2=pi*cos(pi*t);
ddr2=-pi^2*sin(pi*t);

x1=u(3);
x2=u(4);
x3=u(5);
x4=u(6);

e1=x1-R1;
e2=x3-R2;
de1=x2-dr1;
de2=x4-dr2;

f=max([1,norm([e1,e2]),norm([de1,de2])]);

gama=1.5;
y1=de1+gama*e1;
y2=de2+gama*e2;

r1=250;r2=250;

sys(1)=r1*f*norm([y1,y2]);
sys(2)=r2*x(2);

function sys=mdlOutputs(t,x,u)
R1=cos(pi*t);
dr1=-pi*sin(pi*t);
ddr1=-pi^2*cos(pi*t);
R2=sin(pi*t);
dr2=pi*cos(pi*t);
ddr2=-pi^2*sin(pi*t);
```

```
x1=u(3);
x2=u(4);
x3=u(5);
x4=u(6);

e1=x1-R1;
e2=x3-R2;
de1=x2-dr1;
de2=x4-dr2;

v=13.33;
q1=8.98;
q2=8.75;
g=9.8;

M=[v+q1+2*q2*cos(x3) q1+q2*cos(x3);
   q1+q2*cos(x3) q1];

C=[-q2*x4*sin(x3)  -q2*(x2+x4)*sin(x3);
    q2*x2*sin(x3)  0];

G=[15*g*cos(x1)+8.75*g*cos(x1+x3);
   8.75*g*cos(x1+x3)];

f=max([1,norm([e1,e2]),norm([de1,de2])]);

gama=1.5;
y1=de1+gama*e1;
y2=de2+gama*e2;
u1=-(x(1)*f)^2*y1/(x(1)*f*norm([y1 y2])+x(2)^2+0.0000001);
u2=-(x(1)*f)^2*y2/(x(1)*f*norm([y1 y2])+x(2)^2+0.0000001);

dr=[dr1;dr2]-gama*[e1;e2];
ddr=[ddr1;ddr2]-gama*[de1;de2];

fy=M*ddr+C*dr+G;

Kp1=[180,0;0,190];
Kp2=[150,0;0,150];
Kv1=[180,0;0,180];
Kv2=[150,0;0,150];
```

```
alfa1=1;alfa2=1;
beta1=1;beta2=1;

tol(1)=-(Kp1(1,1)+Kp2(1,1)/(alfa1+abs(e1)))*e1-(Kv1(1,1)+Kv2(1,1)/(beta1+
abs(de1)))*de1+fy(1)+u1;
tol(2)=-(Kp1(2,2)+Kp2(2,2)/(alfa2+abs(e2)))*e2-(Kv1(2,2)+Kv2(2,2)/(beta2+
abs(de2)))*de2+fy(2)+u2;

sys(1)=tol(1);
sys(2)=tol(2);
sys(3)=x(1);
```

被控对象子程序：chap10_5plant.m，与仿真方法一中的被控对象相同。

第 11 章　PID 实时控制的 C++ 语言设计及应用

11.1　M 语言的 C++转化

仿真实例

对象采用三阶传递函数为：

$$G(s)=\frac{523500}{s^3+87.35s^2+10470s}$$

采样时间为 1ms，被控对象可离散化为：

$$y(k)=-\mathrm{den}(2)y(k-1)-\mathrm{den}(3)y(k-2)-\mathrm{den}(4)y(k-3)+\mathrm{num}(2)u(k-1)+\mathrm{num}(3)u(k-2)+\mathrm{num}(4)u(k-3)$$

取输入指令为阶跃信号，进行位置跟踪的仿真。PID 控制参数取 $k_p=25$、$k_d=0$，$k_i=0.28$。M 语言的仿真程序为 chap11_1.m，PID 的阶跃响应如图 11-1 所示。转化的 C++语言（Borland C++）程序为 chap11_2.cpp。

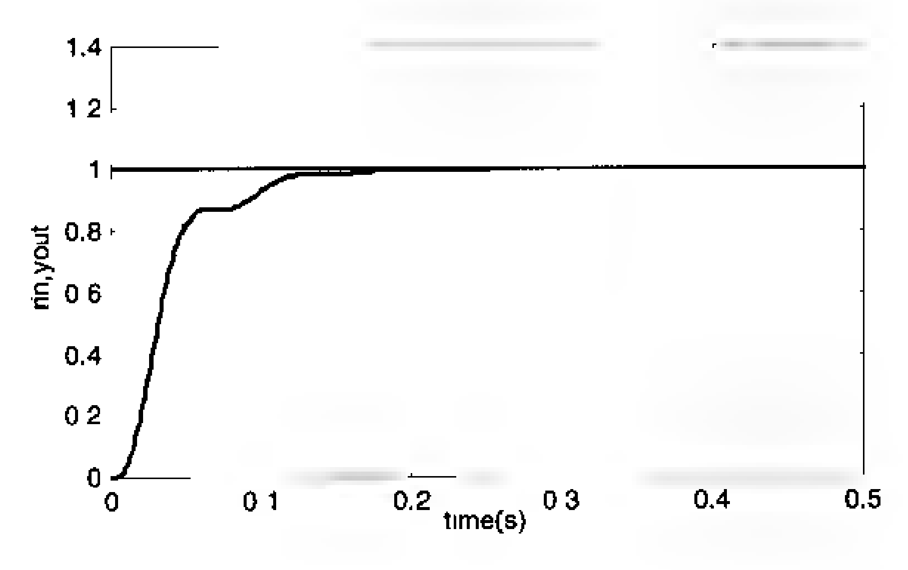


图 11-1　PID 的阶跃响应

M 语言仿真程序：chap11_1.m。

```
%PID Controller
clear all;
close all;

ts=0.001;
```

```
sys=tf(5.235e005,[1,87.35,1.047e004,0]);
dsys=c2d(sys,ts,'z');
[num,den]=tfdata(dsys,'v');

u_1=0.0;u_2=0.0;u_3=0.0;
y_1=0.0;y_2=0.0;y_3=0.0;
x=[0,0,0]';
error_1=0;

for k=1:1:500
time(k)=k*ts;

rin(k)=1;                    %Tracing Step Signal
kp=0.50;ki=0.001;kd=0.001;

u(k)=kp*x(1)+kd*x(2)+ki*x(3);   %PID Controller

%Linear model
yout(k)=-den(2)*y_1-den(3)*y_2-den(4)*y_3+num(2)*u_1+num(3)*u_2+num(4)*u_3;

error(k)=rin(k)-yout(k);

%Return of parameters
u_3=u_2;u_2=u_1;u_1=u(k);
y_3=y_2;y_2=y_1;y_1=yout(k);

x(1)=error(k);                  %Calculating P
x(2)=(error(k)-error_1)/ts;     %Calculating D
x(3)=x(3)+error(k)*ts;          %Calculating I
xi(k)=x(3);

error_1=error(k);
end
figure(1);
plot(time,rin,'b',time,yout,'r');
xlabel('time(s)');ylabel('rin,yout');
```

C++语言仿真程序：chap11_2.cpp。

```
//PID Control
#include <math.h>
int k;
```

```
double yout,u,rin,u_1=0,u_2=0,u_3=0,yout_1=0,yout_2=0,yout_3=0;
double Error_1=0.0,Error_2=0.0;
double ts=0.001;
double timek,pi=3.1415926;
double kp=0.50,ki=0.001,kd=0.001;
double Error,Perror,Ierror,Derror;

void main()
{
for(k=1;k<=5000;k++)
{
        timek=k*ts;
        rin=1;

//Practical Position Degree Signal at time t
yout=2.9063*yout_1-2.8227*yout_2+0.9164*yout_3+0.0853*0.001*u_1+0.3338*0.001
*u_2+0.0817*0.001*u_3;

        Error=yout-rin;
//PID Controller
        Perror=Error;                    //Getting P
        Ierror=Ierror+Error*ts;          //Getting I
        Derror=(Error-Error_1)/ts;       //Getting D

        u=kp*Perror+ki*Ierror+kd*Derror;  //PID Controller

//Update Parameters
        Error_2=Error_1;
        Error_1=Error;

        yout_3=yout_2;
        yout_2=yout_1;
        yout_1=yout;   //Positional Signal at k-1
        u_3=u_2;
        u_2=u_1;
        u_1=u;
        }
        }
```

11.2 基于C++的三轴飞行模拟转台伺服系统PID实时控制

11.2.1 控制系统构成

三轴电动飞行模拟转台伺服系统由机械台体、电控柜两部分组成，如图 11-2 所示。机械台体是三轴仿真系统的主体，由基座和三个运动框架组成，用以安装试验负载，模拟飞行器的三维姿态运动；电控柜由转台系统的测控、显示、电子伺服装置和控制计算机组成，用来构成转台的伺服控制系统，实施对转台的检测和控制。

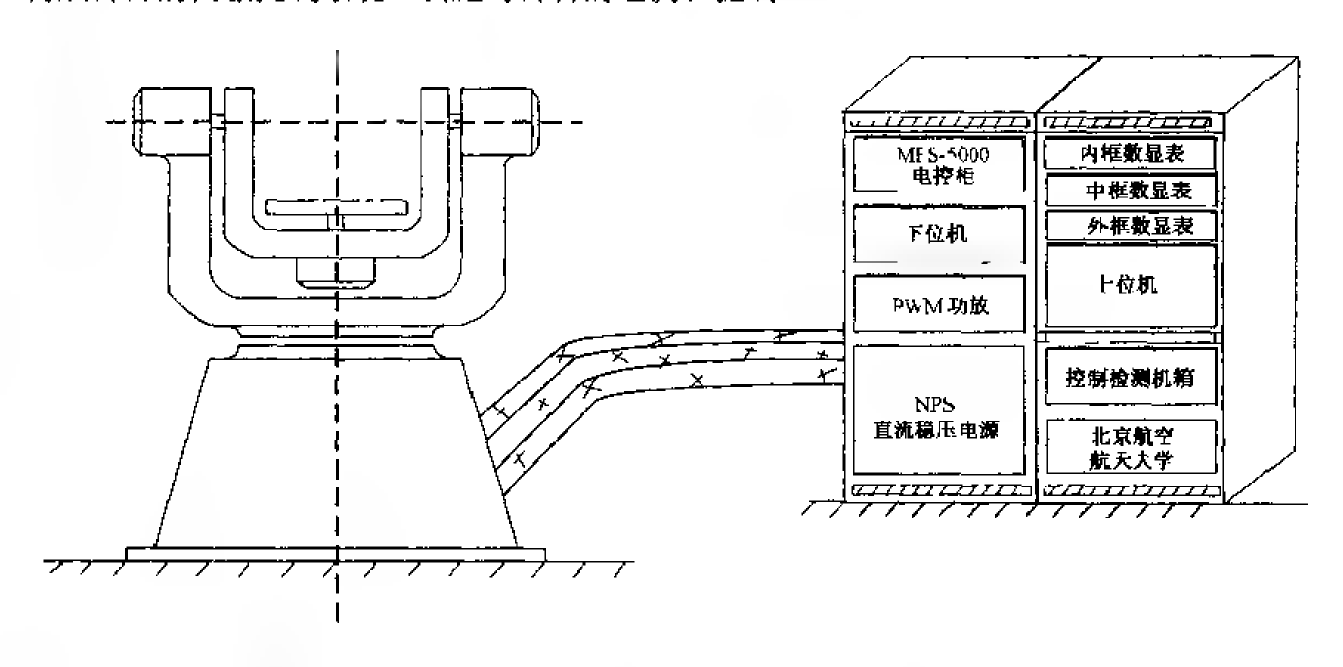


图 11-2 转台系统总体结构图

内、中、外框三个通道的数字伺服控制系统，其原理结构完全相同，如图 11-3 所示。系统采用了微机控制下的脉冲调宽功放装置（PWM），直流力矩电机直接驱动转台框架的数字伺服调速体制。高灵敏度的直流测速机直接构成连续式速度控制回路，有利于系统刚度的提高和频带的拓宽。由高精度测角元件圆感应同步器和数字变换装置构成数字式角位置反馈回路，可以满足系统的精度和性能要求。采用工业控制机作为该伺服系统的主控计算机，能够保证系统快速性能的实现，同时也能很好地完成系统控制律的形成、回路内部的数据采集处理、故障诊断和监控管理，使系统的动态性能与安全可靠性都得到充分的保证。

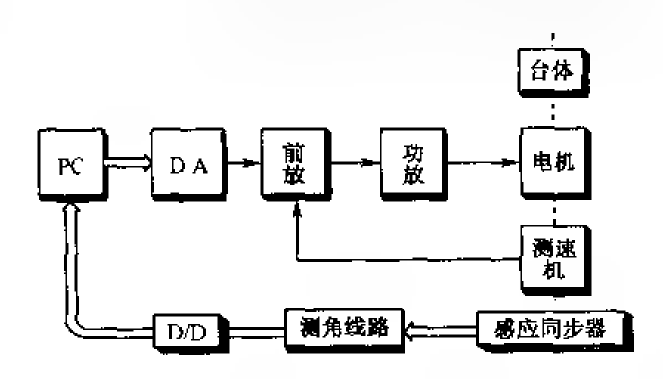


图 11-3 内、中、外三通道数字伺服控制系统原理结构

11.2.2 实时控制程序分析

主程序流程图如图 11-4 所示。

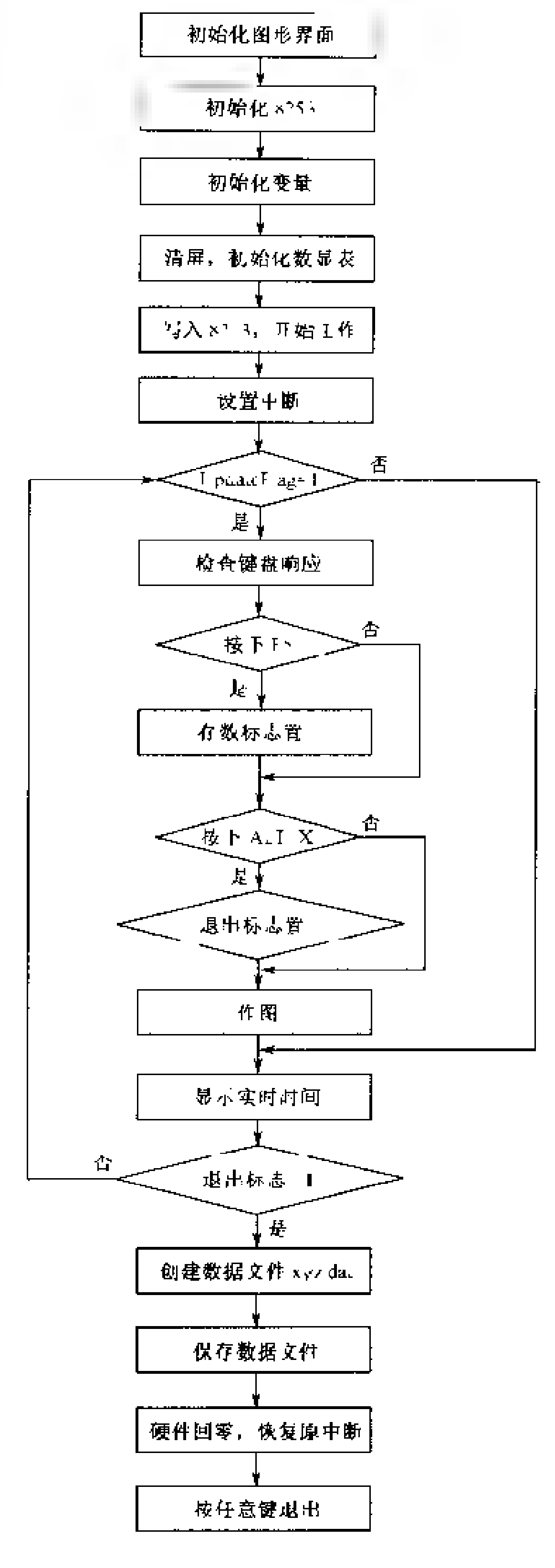


图 11-4 主程序流程图

中断流程图如图 11-5 所示。

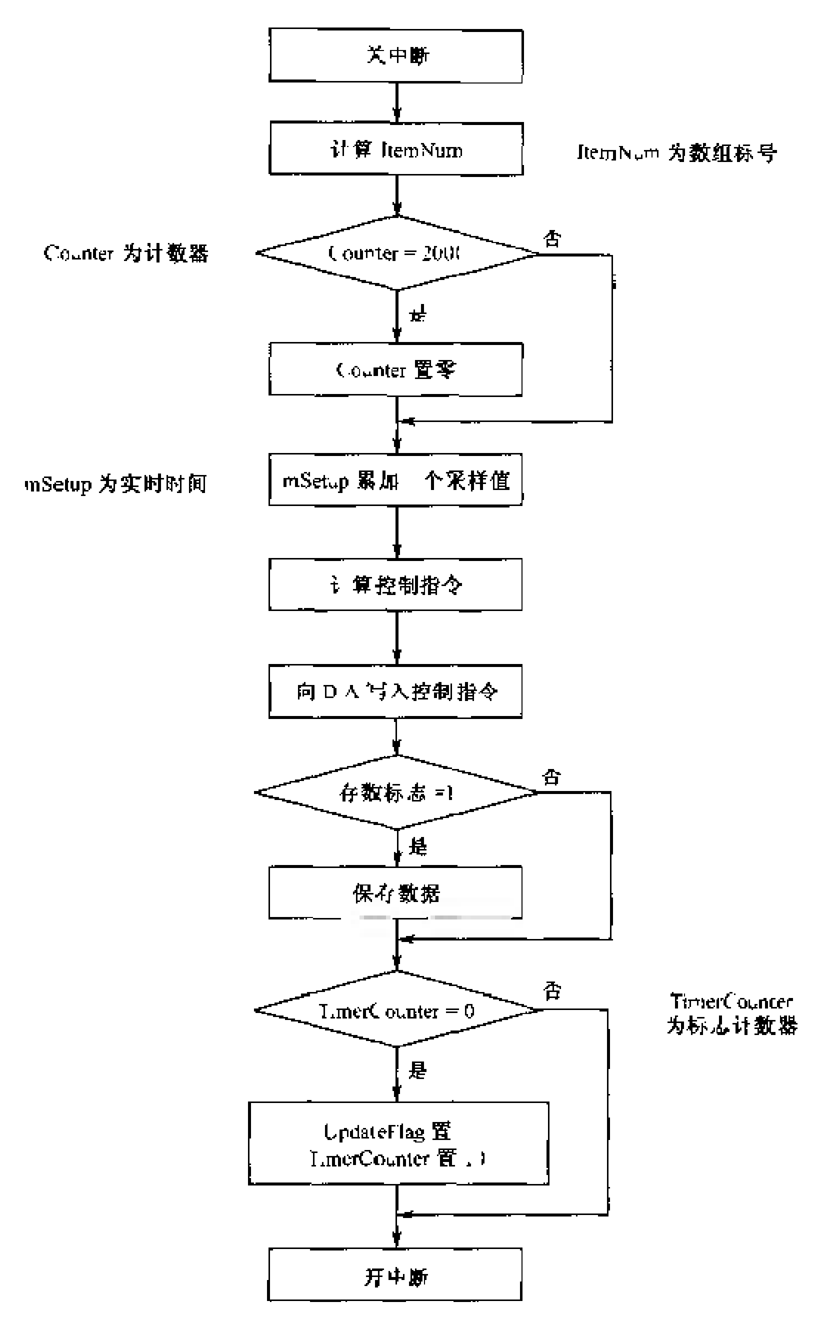


图 11.5 中断流程图

1. 程序中变量及宏定义说明

A 和 *F* 分别表示输入正弦指令的幅值和频率。channel 用来设定控制对象，当 channel＝0 时，被控对象为内框；当 channel＝1 时，被控对象为中框；当 channel＝2 时，被控对象为外框。当 channel 变化时，改变 D/A 板和 D/D 板的地址，从而达到改变控制对象的目的。

TIMER_BASE 为 8253 定时器的地址，TIMER_RATIO 定义为 10，表示每次画图画 10 个点，TIMER_CYCLE 与 TIMER_RATIO 的比值为采样周期。

TIMER_VALUE 用来定义 8253 定时器的初始计数值，当 $N=1$ 时，8253 定时器每隔 1ms 发一次信号，也即采样周期为 1ms；当 $N=10$ 时，8253 定时器每隔 10ms 发一次信号，也即采样周期为 10ms.

Irqnumber 为中断服务程序对应的中断向量号。

DATA_DIMENTION 表示用于存数的数组的维数，DATA_LENGTH 表示存数数组的长

度，可调整它们改变存数的数量。

T8253_MODE_0～T8253_MODE_5 用来表示 8253 的 5 种计数方式，控制程序中实际使用的是 T8253_MODE_3，在该种方式下，8253 计数器产生周期性的方波信号，每隔一个采样周期发出一个上升沿信号给 8259 中断计数器，触发中断服务程序。

T8253_CHANNEL_0～T8253_CHANNEL_2 用来表示 8253 计数器中不同的计数寄存器，分别与内框、中框和外框对应。

T8253_BIN_MODE 表示 8253 计数器的计数方法为使用二进制，T8253_BCD_MODE 表示 8253 计数器的计数方法为使用十进制 BCD 码。

T8253_COUNT_LOCK 表示 8253 计数器锁存，T8253_COUNT_LOW 表示 8253 计数器的写入方法为写入低 8 位，T8253_COUNT_HI 表示 8253 计数器的写入方法为写入高 8 位，T8253_LOW_FIRST 表示 8253 计数器的写入方法为先写低 8 位，再写高 8 位。

EOI 为 8259 中断管理器的选通信号。

KB_C_N_F4～KB_C_A_X 表示键盘上字符对应的数值及其转化结果。

当使用的编译器为 C++编译器时，将_CPPARGS 定义为 …，用于中断服务程序。

2. 整体设计思路

采样时间为 1ms，即每隔 0.001s 执行一次中断，在中断中完成控制律的计算、采集实时输出信号、发出实时控制信号；若存数标志位为 1 时，将每次采样时的输入信号和实时位置信号存入存数数组；每隔 0.01s 将画图标志位置为 1。

主程序中执行一个循环，循环的跳出条件是跳出标志位为 1。在循环中，当画图标志位被置为 1 时，执行画图函数和键盘响应函数，也即每隔 0.01s 画一次图，检查一次键盘响应。当检查到 F5 时，将存数标志位置为 1，这样在接下来的中断中可以执行存数操作；当检查到 ALT＋X 时，将跳出标志位置 1，这时将跳出循环。循环中一直执行显示实时时间的功能。

将存数数组中的数据存入名为 xyz.dat 的文件中，其中第一列位输入指令信号，第二列位输出信号。

3. 子函数说明

- ResetShuxianbiao()

实现数显表的初始化，将内框、中框和外框数显表全部清零，通过向数显表端口写入相应的信号实现。

- ReadD_D()

采用 D/D 板实现位置信号的采集。将数显表产生的 23 位传感信号读入计算机，并转化为相应的实际位置值，连续读入两次数显表的输出值，当它们相等时，认为信号稳定。读入信号与实际角位置之间的比例系数为 0.9，乘以 0.9 后的信号单位为角秒，要再除以 3600 转化为角度。23 位传感信号的第 23 位是符号位，当其为 1 时，表示该实际角位置信号为负值。该函数的返回值为实际角度。

- Write_DA()

将控制信号发送到 D/A 板，实现控制器的输出。

- KbGetKey()和 SetupKeyReaction()

实现键盘管理，键盘参数在头文件 chap11_3.h 中设定。判断按键是否是 F5 或 ALT＋X，

如果是F5，就将存数标志位置1，如果是ALT+X，就将跳出标志位置1。数据保存由F5键实现，退出由ALT+X键实现。

● DataSavedRoutine()

实现数据存储功能。按F5键则保存数据，保存文件名为xyz.dat。此时存数标志位是1，执行存数操作，将指令信号和实际输出信号存入一位数组，由数组长度决定存数多少，存满为止。

● Control()

调用ReadD_D()读入实际位置数据，根据指令信号算出误差，利用PID控制算法算出控制指令。在该函数中可以实现各种控制算法。

● DynamicDisplay()

绘制实时控制曲线，每隔10个采样周期执行一次，每次绘制10个点。界面采取方框的形式，由line()函数实现，每五个点共用一个横坐标，每隔20000个点清屏一次。

中断响应是实时控制中最重要的部分。在本程序中，中断响应采用函数IniTimer()、interrupt()和IrqHook()来实现，中断时间为1ms，在中断时间内完成控制算法及绘图等功能，采用定时器8253模式来实现中断，中断控制器采用8259A模式。中断参数在头文件chap11_3.h中设定。分别介绍如下：

● IniTimer()

实现中断初始化。向8253定时器中写入控制字，采用工作方式3，二进制计数，选用计数寄存器0，先写低8位再写高8位。CPU时钟频率为1.193MHz。

● TimerIrqVect()

中断服务程序，每个采样周期执行一次，完成的任务为：计算实时时间；调用Control()实现控制律；调用Write_DA()将控制信号送出；调用DataSavedRoutine()存数；每隔10个采样周期将画图标志位置1，发8259中断管理器选通信号。

● IrqHook()

实现中断地址挂接，即保存旧的中断向量，设置新的中断向量。

● ReleaseHardware()

恢复旧的中断向量，调用Write_DA()输出零控制信号，实现控制电机的清零。

11.2.3 仿真程序及分析

仿真实例

转台中框为二阶传递函数：

$$G(s)=\frac{1770}{s^2+60s+1770}$$

采样时间为1ms，离散化为：

$$y(k)=1.94y(k-1)-0.94y(k-2)+0.0008674u(k-1)+0.0008503u(k-2)$$

channel和Signal为选择输入信号类型变量，其中channel=1为转台中框，Signal=1为正弦输入信号，其幅值为0.010，频率为0.50Hz。采用PD进行位置跟踪的实时控制，其中$k_p=10, k_d=1.0$。当$M=1$时，位置信号由D/D板采集而得，为PID实时控制；当$M=2$时，位置信号由传递函数给出，为PID仿真控制。位置跟踪结果实时控制如图11-6所示。

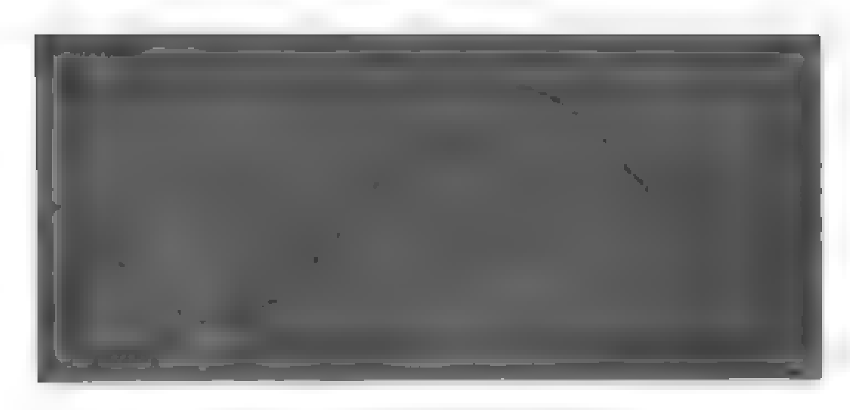


图11-6 位置跟踪实时控制结果

仿真程序：chap11_3.cpp.

```
[illegible]
[illegible]
[illegible] directory/
[illegible]
[illegible] c:\borlandc\inc[illegible]
#include <math.h>
#include <conio.h>          [illegible]ing clrscr() and get[illegible]
#include <graphics.h>
#include <std[illegible]>
#include <dos.h>            [illegible] and _enable()
#include <string.h>
#include [illegible]           [illegible]key[illegible]
#include [illegible]           [illegible]
#include <chap[illegible]>      [illegible] parameters

#define [illegible]
#define INPORT  inportb
#define T        500
#define DD PORT 0x380    //D/D board base address

int channel=1;           //three frame: 0:inner; [illegible]
[illegible]
int Signal=2;            //Step signal

double PositionCommand[TIMER_RATIO
double Line[illegible]
double Curre[illegible]
double [illegible]

double [illegible]
double [illegible]
```

```
double pi 3.14159265358979;
double timezt-0;
int flag-0,
unsigned short    UpdateFlag-0;
double mStep,mStep 1,Step,SaveStep-0;   ,mStep 1 t)-mStep(t 10)
//mStep must defined double
unsigned short    ItemNum;
unsigned short    TimerCount;
unsigned long     Counter;

unsigned short    flagF5;
unsigned short    FinishSimulate;
FILE *xyz;

#define DD PORT  0x380   / Inner frame
void ResetShuXianBiao()
{
    outport(DD_PORT,0x0,;
    outport(DD_PORT,0xf,;
    outport DD_PORT+4,0x0);
    outport DD_PORT+4,0xf ;
    outport(DD_PORT+8,0x0),
    outport DD_PORT+8,0xf);
}

double angle_1;
double ReadD_D(unsigned short channel
{
    unsigned short Data H1,Data_L1,Data_H2,Data_L2,
    long int data,angle_degree,angle minute,angle second,
    double angle,angle1,errory;
    do {
        Data L1-inport(DD PORT+4*channel ;
        Data_H1 inport(DD PORT+2+4*channel);
        Data_L2 inport DD_PORT+4*channel,;
        Data_H2 inport DD PORT+2+4*channel);
       }while (Data_L1. Data_L2   (Data_H1. Data_H2),;

     data- (Data_H1 & 0x3f)*65536+Data L1 ;
     angle1 data*0 9;         Second per data
     if( Data H1 & 0x80  --0x80   //nagative if Data_H.7 .-1
        angle1 -angle1;
```

```
    angle=angle1/3600.0;   //Change from Second to degree
    return(angle);
}

#define Da_Board  0x1A0
void Write_DA(double dValue, int channel)
{
   unsigned int hi,low,hilow;
   int state;
   double voltage;
   voltage=dValue;
   if(voltage>2.0)voltage=2.0;
   if(voltage<-2.0)voltage=-2.0;
   hilow=(unsigned int)((voltage)*65535/20);

   hi=hilow&0xff00;
   hi=hi>>8;
   low=hilow&0xff;

   outportb(Da_Board+0,channel); //Select channel
   do
     {
      state=(inportb(Da_Board+0))&0x80;
     } while(state==1);    //read bit D7; if D7=0 write data

   outportb(Da_Board+1,low);   //write low byte
   outportb(Da_Board+2,hi);    //write high byte
   outportb(Da_Board+3,0);     //start da
}

int KbGetKey(int *ScanCode)
{
     int Key;
     int KeyCode;
     Key=bioskey(0);
     if((Key&0x00ff)==0)
     {
      KeyCode=0;
      *ScanCode=(Key>>8)&127;
     }
     else
     {
```

```
        KeyCode=Key&0xff;
        *ScanCode=0;
      }
      return(KeyCode);
}

int SavedFlag,SaveCounter;
void SetupKeyReaction(void)
{
      int Key,Scan;
      if(kbhit())
      {
        Key=KbGetKey(&Scan);
        if (Key==KB_C_N_F5 && Scan==KB_S_N_F5)
        {
          SavedFlag=1;
          SaveCounter=0;
        }
        if(Key==KB_C_A_X && Scan==KB_S_A_X)
        {
          FinishSimulate=1;
        }
      }
}

double huge DataSaved[DATA_DIMENTION][DATA_LENGTH];
void DataSaveRoutine( long int SaveSpan)
{
      if(SavedFlag==1)
        {
          if((SaveCounter<=SaveSpan*DATA_LENGTH)) //DATA_LENGTH is Defined by
chap11 .h
           {
           int SaveIndex=SaveCounter/SaveSpan;
   //Save data(rin,yout) to xyz.dat
           DataSaved[0][SaveIndex]=PositionCommand[ItemNum];
           DataSaved[1][SaveIndex]=CurrentPosition[ItemNum];
           }
          SaveCounter++;
          if(SaveCounter>=SaveSpan*DATA_LENGTH) { SavedFlag=0; SaveCounter=0; }
        }
       if(SavedFlag==0) SaveCounter=0;
```

```
{
int M;
double yout,error,derror;
double u_1=0,u_2=0,y_1=0,y_2=0,error_1=0,error_2=0,ei=0;
double Control(double rin,unsigned short channel)
{
M=2;
if(M==1)     //Realtime control
{
yout=ReadDA(channel);    //Read realtime data
CurrentPosition[ItemNum]=yout;
}
if(M==2)    //Simulation control
{
yout=1.94*y_1-0.94*y_2+0.0008674*u_1+0.0008503*u_2;
}

CurrentPosition[ItemNum]=yout;
yout=CurrentPosition[ItemNum];

F=0.50;
A=0.010; if(A<0.0) A=0.0001;
rin=A*sin(F*2*pi*timezt);
error=rin-yout;

u=10.0*error+1.0*derror+0*ei;
//Update Parameters
y_2=y_1;y_1=yout;
u_2=u_1;u_1=u;
error_2=error_1;
error_1=error;

return u;
}
int Cycles=5;  //Display cycle times
void DynamicDisplay()
{
    //Make a window
    char   strA[50];
    int    i,Spoint;
    int    bottom,middle,top,right,left;
```

```
    bottom=300;middle=200;top=100;left=50;right=550;
    setcolor(RED);
    outtextxy(270,50,"PID Controller");
    line(left,top,left,bottom);        //lineleft
    sprintf(strA,"%f",A*6.0/5.0);
    outtextxy(36,top-10,strA);
    line(left,top,right,top);          //linetop
    outtextxy(36,middle,"0");
    line(left,middle,right,middle);     //linemiddle
    sprintf(strA,"%f",-A*6.0/5.0);
    outtextxy(36,bottom+5,strA);
    line(left,bottom,right,bottom);     //linedown
    line(right,top,right,bottom);       //lineright

//Make Curve Range
    for(i=0;i<TIMER_RATIO;i++)   //Plot 10 points once a time
    {
     Spoint=mStep_1/Step TIMER_RATIO+i;  //Start point

     Spoint Spoint/Cycles;  //20000/(10*T)=One Cycle
     if(Spoint%T==0)
     {
        clrscr();
     }
     putpixel(Spoint-(Spoint/T)*T+50, 100*5.0/6.0*
             (Line1[i]/A)+200,BLUE);    //Practical output
     putpixel(Spoint-(Spoint/T)*T+50,-100*5.0/6.0*
             (Line2[i]/A)+200,RED);     //Ideal output
     putpixel(Spoint-(Spoint/T)*T+50,-100*5.0/6.0*
              u+200,BLACK);             //Control output
    }
}

void IniTimer( unsigned char timer_mode ) //Initial 8253
{
    unsigned char value;
    _disable();

//Setting control word
    OUTPORT(TIMER_BASE+3,timer_mode);
    value = TIMER_VALUE ;         //Low 8 bits
    OUTPORT( TIMER_BASE , value );
```

```
    val.e - TIMER VALUE   256 ,  ./High 8 bits
    OUTPORT, TIMER_BASE , value ,;
     enable(,;
}

void interrupt ( *OldIrqVect ), __CPPARGS  ;
void interrupt TimerIrqVect, __CPPARGS )     Intcr ipt processing
{
    .nsigned short 1;
     disable();
    ItemNum   Co.nter%TIMER_RATIO;

   1f .Co.nter%20000- 0
    {
     Co.nt er 0;
     SaveStep-SaveStep+mStep;
     mStep-0;
    }
    mStep +  Step;           k*ts: TIMER_CYCLE   TIMER_RATIO;

1f  Signal- 1,   , Dynamic Signal
{   PositionCommand[ItemNum] A*sin(F*2*p.*mStep ;  )
1f .Signal -2)   / Static Signal
{   PositionCommand[ItemN.m]-A;  }

    u-Control PositionCommand[ItemNum],channel ;

    Write DA(u,channel,;
    DataSaveRoutine 1 ;

    Counter++;
    1f( --TimerCo.nt)  0     Updating 1 times while interupted by 10 times

      UpdateFlag 1;
      mStep 1-mStep;
      TimerCount-TIMER_RATIO;
     }
     OUTPORT(EOI,0x20,;   OCW2 value:0010 0000
    _enable  ;
 ,

void IrqHook,unsigned short irqnumber, void interrupt ( *newVect , ( __CPPARGS ) )
```

```
{
     _disable();
     OldIrqVect = getvect ( irqnumber );
     setvect ( irqnumber , newVect );
     _enable();
}

void ReleaseHardware()
{
     _disable();
     setvect (Irqnumber,OldIrqVect);
     Write_DA(0.0,0);
     Write_DA(0.0,1);
     Write_DA(0.0,2);
     _enable();
}

main(void)
{
     int i,j,k;
     unsigned char timer0_mode,strtime[10];
     int driver,mode;
     driver=DETECT;
     initgraph(&driver,&mode,"");
     timer0_mode=T8253_MODE_3|T8253_CHANNEL_0|
          T8253_BIN_MODE|T8253_LOW_FIRST;  //8253 timer setting

     Step=TIMER_CYCLE/TIMER_RATIO;         //Step=0.01/10=0.001s=ts
     mStep=0;
     mStep_1=0;
     Counter=0;
     TimerCount=TIMER_RATIO;

     clrscr();
     ResetShuXianBiao();
//Using interupt function
     IniTimer(timer0_mode);  //8253 timer setting
     IrqHook(Irqnumber,TimerIrqVect);

     while(1)              //1 is true
     {
      timert=SaveStep*mStep;
```

```
    if(UpdateFlag==1)
    {
       UpdateFlag=0;
       for(i=0;i<TIMER_RATIO;i++)
       {
        Line1[i]=PositionCommand[i];
        Line2[i]=CurrentPosition[i];
        }
       SetupKeyReaction();
       DynamicDisplay();
     }

    sprintf(strtime,"%f",timezt);
    bar(260,380,360,360);
    outtextxy(285,385,"time(s)");
    outtextxy(280,370,strtime);

//  if(kbhit()) { break; } //Break by any key
    if(FinishSimulate==1)  //Break by "ALT+X"
     break;
   }//End of for() loop

//Open xyz.dat for save data
  if ((xyz = fopen("xyz.dat", "w"))== NULL)
     {
    printf("Cannot open output file xyz.dat.\n");
    exit( 1 );
     }
//xyz=fopen("xyz.dat","w");

  for( i=0;i<DATA_LENGTH;i++)
    {
    for( j=0; j<DATA_DIMENTION; j++)
        {
         fprintf(xyz," %10.6f ",DataSaved[j][i]);
        }
    fprintf(xyz,"\n");
    }
fclose(xyz);

ReleaseHardware();  //Clear all output u
getch();  //Any key to exit
```

```
closegraph();
//Restores the original video mode detected by initgraph
restorecrtmode();
return 0;
} //End of main()
```

参数初始化程序：**chap11_3.h**。

```
#define TIMER_BASE      0x40    //Timer base address
#define TIMER_RATIO     10      //Display timer rate
#define TIMER_CYCLE     0.001*TIMER_RATIO   //Display time cycle

#define TIMER_VALUE 1193   //Define interupt time:sampling time(ts 0.001s)

#define Irqnumber 0x08    //Realtime interupt kind:Clock Interupt

//Save data parameters
#define DATA_DIMENTION        2
#define DATA_LENGTH           100

//8253 timer
#define T8253_MODE_0         0x00
#define T8253_MODE_1         0x02
#define T8253_MODE_2         0x04
#define T8253_MODE_3         0x06
#define T8253_MODE_4         0x08
#define T8253_MODE_5         0x0a

#define T8253_CHANNEL_0      0x00
#define T8253_CHANNEL_1      0x40
#define T8253_CHANNEL_2      0x80

#define T8253_BIN_MODE       0x00
#define T8253_BCD_MODE       0x01

#define T8253_COUNT_LOCK     0x00
#define T8253_COUNT_LOW      0x10
#define T8253_COUNT_HI       0x20
#define T8253_LOW_FIRST      0x30

//8259 control(P197)
#define EOI           0x20
```

```
//Define key value//
#define KB_C_N_F4 0
#define KB_S_N_F4 62
#define KB_C_N_F5 0
#define KB_S_N_F5 63
#define KB_S_A_X 45
#define KB_C_A_X 0

#ifdef __cplusplus
    #define __CPPARGS ...
#endif
```

参考文献

1 薛定宇．控制系统计算机辅助设计．北京：清华大学出版社，1996

2 傅信鑑．过程计算机控制系统．西安：西北工业大学出版社，1995

3 陶永华，尹怡欣，葛芦生．新型PID控制系统及其应用．北京：机械工业出版社，1998

4 薛定宇，陈阳泉．基于Matlab/Simulink的系统仿真技术与应用．北京：清华大学出版社，2002

5 韩京清，袁露林．跟踪微分器的离散形式．系统科学与数学，1999，19(3):268~273

6 王耀南．智能控制系统．长沙：湖南大学出版社，1996

7 诸静．模糊控制原理与应用．北京：机械工业出版社，1999

8 Albus J. S., A new approach to manipulator control: the cerebellar model articulation controller(CMAC). Journal of Dynamic Systems. Measurement and Control，1975，97，220~227

9 Commuri S., Lewis F L., CMAC Networks for Control of Nonlinear Dynamic System： Structure, Stability and Passivity. Automatica, 1997，33(4)：635~641

10 徐丽娜．神经网络控制．哈尔滨：哈尔滨工业大学出版社，1998

11 蒋志明，林延圻，黄先祥．基于CMAC的带有未知负载干扰电液位置伺服系统的自学习控制．控制与决策，2000，5，15(3)：368~370

12 汪雷，周国兴，吴启迪．基于Hopfield神经网络的直流传动系统模型参考自适应控制．电工电能新技术，2000，2：11~16

13 F. J. Lin. R. J. Wai, C. C. Lee. Fuzzy neural network position controller for ultrasonic motor drive using push-pull DC-DC converter. IEE Proc.-Control Theory Appl., 1999, 146(1): 99~107

14 陈国良，王煦法，庄镇泉，王东生．遗传算法及其应用．北京：人民邮电出版社，1996

15 周明，孙树栋．遗传算法原理及应用．北京：国防工业出版社，1999

16 Leether Yao, William A. Sethares. Nonlinear Parameter Estimation Via the Generic Algorithm. IEEE Transactions on Signal Processing, 1994, 42(4)：927~935

17 Wen-hua chen, Donald J. Balance, Peter J. Gawthrop, John O'Reily. A Nonlinear Disturbance Observer for Robotic Manipulators. IEEE Transactions on Industrial Electronics, 2000, 47(4)：932~938

18 Carl J. Kempf, Seiichi Kobayashi. Disturbance Observer and Feedforward Design for a High-speed Direct-Drive Positioning Table. IEEE Transactions on control system technology，1999，7(5)：513~527

19 陈剑桥．非线性PID控制器的计算机辅助设计．扬州职业大学学报，2001,5(4):12~15

20 Ho Seong Lee, Masayoshi Tomizuka. Robust Motion Control Design for High-Accuracy Positioning System. IEEE Transaction Industrial Electronics，1996，43(1)：48~55

21 黄文梅，杨勇，熊桂林．系统分析与设计—MATLAB语言及应用．长沙：国防科技大学出版社，2001

22 肖永利，张琛．位置伺服系统的一类非线性PID调节器设计．电气自动化，2000，1：20~22

23 Manayathara T. J., Tsao T. C., Bentsman J. N, Ross D., Rejection of Unknown Periodic Load Disturbances in Continuous Steel Casting Process Using Learning Repetitive Control Approach . IEEE Transactions on Control System Technology, 1996, 4(3)：259~265

24 T. E. Peery, H.Ozbay. H^{∞} Optimal Repetitive Controller Design for Stable Plants. Journal of Dynamic Systems. Measurement and Control，1997，119：541~547

25 M. Tomizuka. Zero Phase Error Tracking Algorithm for Digital Control. ASME Journal of Dynamic Systems, Measurement, and Control 1987, 109: 65~68

26 D. Torfs, et. al. Extended Bandwidth Zero Phase Error Tracking Control of Nonminimal Phase Systems. ASME Journal of Dynamic Systems, Measurement and Control, 1992, 114: 347~351

27 J. Z. Xia, C. H. Menq. Precision Tracking Control of Non-minimum Phase Systems with Zero Phase Error. International Journal of Control, 1995, 61: 791~807

28 施国强. 吊车-双摆计算机控制系统. 北京航空航天大学学士学位论文，2001.6

29 A. S. Hodel, C. E. Hall. Variable-structure PID Control to prevent integrator windup. IEEE Transactions on Industrial Electronics, 2001, 48(2): 442~451

30 邓聚龙. 灰色控制系统. 武汉：华中理工大学出版社，1985

31 邓聚龙. 灰色预测与决策. 武汉：华中理工大学出版社，1988

32 刘思峰，郭天榜，党耀国. 灰色系统理论及其应用. 北京：科学出版社，1999

33 B. Armstrong, P. Dupont, C. Canudas de Wit. A Survey of Models, Analysis Tools and Compensation Methods for the Control of Machines with Friction. Automatica, 1994,30(7)：1083~1138

34 C. Canudas de Wit, H. Olsson, K. J.Astrom, P. Lischinsk. A new model for control of systems with friction. IEEE trans. Automatic Control, 1995，40(3)：419~425

35 Karnopp D. Comput. Computer Simulation of Stick-slip Friction in Mechanical Dynamic Systems. Journal of Dynamic Systems, Measurement and Control, 1985, 107：100~103

36 尔联洁. 自动控制系统. 北京：航空工业出版社，1994

37 冯国楠. 现代伺服系统的分析与设计. 北京：机械工业出版社，1990

38 胡祐德，曾乐生，马东升. 伺服系统原理与设计. 北京：北京理工大学出版社，1993

39 徐宁寿. 随机信号估计与系统控制. 北京：北京工业大学出版社，2001

40 申铁龙. 机器人鲁棒控制基础. 北京：清华大学出版社，2000

41 陈启军，王月娟，陈辉堂. 基于PD控制的机器人轨迹跟踪性能研究与比较. 控制与决策，2003，18(1)：53~57

42 焦晓红，李运锋，方一鸣，耿秋实. 一种机器人鲁棒自适应控制法. 机器人技术与应用，2002，3：40~43